Surface Topological Analysis for Image Synthesis

A Thesis
Presented to
The Academic Faculty

by

Eugene Zhang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
July 2004

Surface Topological Analysis for Image Synthesis

Professor Greg Turk, Committee Chair

Professor Konstantin Mischaikow

Professor Jarek Rossignac

Professor Andrzej Szymczak

Professor John C. Hart
(University of Illinois)

Date Approved: April 20, 2004

To my family,

both in China and in the United States,

for their love, support and sacrifice.

PREFACE

In these days the angel of topology and the devil of abstract algebra fight for the soul of every individual discipline of mathematics – *Hermann Weyl (1885-1955)*

I first read this quote from a textbook translated into Chinese. At the time, I was a sophomore in college and had just spent the past summer teaching myself modern algebra. I recall my exact thoughts on Weyl's saying. "Why is topology the angel and abstract algebra the devil? And what is topology anyway?". These questions have gone a long way in defining my interests in Mathematics, and later in Computer Graphics.

Fast forward...

On a summer afternoon in 2001, my advisor Greg Turk presented his work on "Texture Synthesis on Surfaces" at a group meeting. In this work, he had developed algorithms for copying interesting patterns from pictures onto 3D surfaces. For this to work, Greg argued, one needs a vector field. The impacts of the vector fields that he had used were obvious in some of the results. At the time, I did not realize the impact of this work on my subsequent research at Georgia Tech. In fact, both topics presented in this thesis were inspired by Greg's work on texture synthesis on surfaces. First, the user can control the synthesis by designing vector fields with desired behaviors. Second, surface parameterization allows the synthesis results to be saved in a texture map for interactive display. In proposing solutions to both problems, I have used several ideas based on topology, the "angel" that I first heard about from Weyl.

ACKNOWLEDGEMENTS

I will always be grateful to my advisor, Greg Turk, for his guidance and encouragement, and for his care and friendship. This thesis would not have been possible without his insight. It was through working with Greg that helped me decide to be a professor after I graduate. I hope I will make him proud one day by helping my own students realize their potentials.

I am indebted to Konstantin Mischaikow for bringing me into the world of dynamical systems. His insight in Conley index theory helps lay a solid foundation for my work on vector field design. I wish to thank Jarek Rossignac for his insight and knowledge about various disciplines, which have helped broaden my view of geometric modeling. Thanks to Andrzej Szymczak for the many long discussions about my research. He has helped me see deeper into several problems and develop new research ideas. I also wish to thank John C. Hart for his valuable suggestions on improving my research and this thesis.

I am grateful for the opportunities to work with John Stasko and Jessica Hodgins during my first year at Georgia Tech. I also appreciate the discussions with Irfan Essa on various research topics.

Polygonal models are necessary for any of my algorithms, and I have spent many hours working with several models that are still fascinating to me even today. I wish to thank the following individuals and groups for the models that they have provided: Mark Levoy and the Stanford Graphics Group for the bunny and dragon models, Zoë Wood, Hughes Hoppe, Mathieu Desbrun and Peter Schrörder for the genus six Buddha, and Cyberware for the Venus model. I have used several tools for geometric processing. For this I wish to thank Michael Garland for his Q-Slim simplification program, Peter Linstrom for his mesh simplification and rendering programs, F.S. Nooruddin for the mesh repair tool, and James Hays and Irfan Essa for their high-quality painterly rendering program.

My research have been financially supported by NSF ACI-0083836 and DMS-0138420. I would like express my appreciation for the sponsors.

I would like to thank my colleagues and the staff members at the GVU Center and the College Computing for their help. I would also like to thank my fellow members of the Animation Lab, the Geometry Group, the CPL group, and the CARGO group. In particular, I would like to thank Gabriel Brostow, Mark Carlson, Huong Quynh Dinh, Vivek Kwatra, Ron Metoyer, Alla Safonova, Rawesak "Tee" Tanawongsuwan, and Brooks van Horn.

I wish to thank my wife, Yue Zhang, for her love and inspirations before and during my stay at Georgia Tech.

I am grateful to my parents' love and motivations over the years. Thanks to my brother and sister-in-law for their financial support when I first came to the United States.

I also wish to thank my other half of the family: my parents-in-law for their love and care, and Ge, Mei and Eric for their help and friendship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

My thesis work is related to solving two problems on mesh surfaces by performing topological analysis. Many graphics applications, such as high-quality and interactive image synthesis, benefit from the solutions to these problems. The two problems that I address are: surface parameterization and vector field design on surfaces.

Surface parameterization refers to segmenting a mesh surface into a number of patches and unfolding these patches onto a plane without any overlap. Surface parameterization is primarily used for storing surface signals into texture maps, which speeds up the subsequent rendering process. One of the most important quality measurements for surface parameterization is stretch. Stretch causes uneven sampling rate across the surface and needs to be avoided whenever possible. My parameterization technique is based on the idea that a surface can be approximated by a collection of relative simple shapes, such as cylinders, cones, flat disks, and spheres. Unfolding them results in relatively little stretch. I decompose the surface by identifying major features contained in the surface such as handles and large protrusions. This is achieved by performing topological analysis of a distance-based function on the surface and locating its local maxima and saddles. Also, I will describe two techniques to reduce stretch during patch unfolding: the Green-Lagrange tensor, and a *virtual boundary*.

Vector field design on surfaces allows a user to create a wide variety of vector fields on mesh surfaces and to have control over the topology of these vector fields. A vector field design system is useful for many graphics applications, such as texture synthesis, non-photorealistic rendering, and fluid simulation. Vector field design is a new and challenging problem. First, a vector field design system should provide control over the geometric and topological characteristics of a vector field. Second, vector field

design for mesh surfaces requires an efficient way of representing and analyzing surface vector fields. In this thesis, I present a vector field design system that allows a user to interactively create a wide variety of vector fields with control over its analytical characteristics, such as the curl and divergence, and its topological characteristics, such as the number and location of the singularities. This is achieved through a set of editing operations provided by the system. In addition, I will describe a new piecewise interpolating scheme for representing continuous vector fields defined on 3D mesh surfaces. This is based on the well-known concepts of *geodesic polar maps* and *parallel transport* from classical differential geometry. Furthermore, I apply the system to several graphics applications: painterly rendering of still images, pencil-sketches of surfaces, and texture synthesis.

CHAPTER I

INTRODUCTION

The main theme of this thesis is to perform *topological analysis* on *mesh surfaces* for image synthesis.

Computer Graphics is concerned with producing images that convey information. Three-dimensional graphics, the branch in Computer Graphics that deals with 3D shapes, has been gaining popularity during the last decade, thanks in part to the development of shape capture systems. Nowadays, high quality and interactive rendering (image synthesis) of 3D surfaces has become one of the most fundamental requirements for 3D graphics applications.

For instance, in special effect and computer game applications, it is often desirable to put a pre-defined texture onto a 3D surface, such as a human's face or an animal model. The texture can contain the lighting and shadow information in a particular environment, or it can be some texture captured from the real world, such as a leopard's spots or a wood's pattern. Other applications may create textures over a 3D surface through simulation, such as a reaction-diffusion process. In Scientific Visualization, people often perform simulation over 3D surfaces, such as weather simulation over a spherical height field and aerodynamic simulation near the surface of an airplane's wing.

While these applications have broadened the applications of Computer Graphics, they also bring about many great challenges. For example, there have been many texture synthesis methods that produce an image. However, it is not straightforward to extend these techniques to curved surfaces due to topological and geometric reasons.

Geometrically, a curved surface is rather different from a plane. For instance, depending on where you are on a surface, the area of a disk with a fixed radius r may differ. Around a flat region, the area is nearly $2\pi r$. However, if you happen to be standing on the tip of a large protrusion, the area of the disk can be arbitrarily small. On the other hand, if you are at the place where the protrusion joins the rest of the surface, the area can be arbitrarily

large. This is a direct result of surface curvature, which measures how curves are bent on the surface. In terms of image synthesis, such inhomogeneous surface geometry means that if you copy a circular image of radius r onto a surface, the image may become distorted where the geometry is rather different from the plane.

Even if we are not concerned with the distortions in geometric measurements, such as distances, angles and areas, there is a more fundamental difference between a plane and a general 3D surface: topology. Imagine tiling the surface of a cube with a squared image. Suppose that the image has an orientation of *north*, *south*, *west*, and *east*. Also suppose that an edge is a *legal* edge in the tiling if it matches a pair of north/south or a pair of east/west between neighboring squares. A tiling is *legal* if every edge is a legal edge. On a plane, this is an easy task. We can divide the plane into regular 2D grids, and assign the texture image to every grid such that its north side coincides with the upper border and its east side coincides with the right border. On the other hand, doing this on a cube is impossible. One may try to divide each face of the cube into a collection of smaller quadrilaterals. However, there will be at least one quadrilateral (quad) which does not have a legal assignment. This is the same problem of trying to name the direction *north* everywhere on the Earth. At the North Pole, no outgoing direction is north. At the South Pole, every outgoing direction is north. Such places are called singularities in a direction field, which are the places where continuity breaks down. In general, the more topological features (handles) a surface has, the more singularities (mainly saddles) it is likely to have.

Topology and geometry are not completely separate either. Back to the problem of tiling a sphere using quads. If we require continuity except at the singularities, we will be able to find a tiling. However, such a tiling will have different texture image tiled at the different scales, large near the equator, and small near the poles. In other words, distortions are a by-product of the continuity requirement on a sphere. If we drop this requirement, then a uniform-scale tiling is easily available. Just divide the sphere into six quadrilaterals as a cube, and then assign the texture image individually to each quad. To summarize, continuity over a curved surface often results in non-uniformity, and uniformity on a curved surface leads to discontinuity.

Problems like these reveal the challenges posed by 3D surfaces that are absent from a plane. There are normally several approaches to handle these problems depending on the applications. For instance, if continuity is not required, the surface can be divided into disjoint regions such that uniformity within each region is maximized. At the same time, discontinuities are allowed along region boundaries. On the other hand, if continuity is required, then the algorithms will seek to evenly distribute distortion across the surface such that the number of places where discontinuity occurs is minimized. When discontinuity is unavoidable, the user should have control over the number, the location, and the form of the discontinuity. In addition, complex geometry often provides places where the discontinuities can be placed to minimize their visual effects.

In this thesis, I present solutions to two problems in 3D graphics, *surface parameterization* and *vector field design on surfaces*. Surface parameterization is primarily used to store surface signals, such as colors and normal vectors, in a planar texture map for subsequent efficient rendering. Vector field design allows a user to create vector fields for various applications, such as non-photorealistic rendering, texture synthesis and fluid simulation, in which an input vector field is necessary. Different vector fields lead to different visual effects. While the two problems seem to have little to do with each other, they share one major attribute: the solutions to these problems are linked to some degree to the topology of the underlying surface. The topology of a closed, orientable surface is determined by the number of handles that are contained in the surface. Several well-known theorems from differential geometry and differential topology link the topology of the surface to the total surface Gaussian curvature (Gauss-Bonnett theorem), the total signed Morse indices of a Morse function (Morse theory), and the total Poincaré indices of any continuous vector field with only isolated singularities (Poincaré-Hopf theorem). These links provide a two-way avenue between the surface and the signals defined on it. First, we gain insight about a surface by performing analysis of some carefully designed surface functions. This is the case for my solution to the surface parameterization problem. By performing topological analysis of some surface distance-based functions, I am able to extract the geometric and topological features contained in the surface. Second, the topology of a surface constrains the topology

of the vector fields defined on it. In my vector field design system, I provide two topological editing operations, singularity pair cancelation and singularity movement, which respect this relationship. Next, I will introduce and discuss the two problems individually.

1.1 Surface Parameterization

1.1.1 The Problem

Surface parameterization is a well-studied problem in Computer Graphics. In general, surface parameterization refers to segmenting a 3D surface into one or more patches and unfolding these patches onto a plane without any overlap. Borrowing terminology from mathematics, this is often referred to as creating an *atlas* of *charts* for a given surface. This process is necessary for many graphics applications in which properties of a 3D surface, such as colors and surface normal, are sampled and stored in a texture map. The quality of the parameterization greatly affects the quality of subsequent applications. One of the most important quality measurements is stretch. When unfolding a surface onto a plane, stretching occurs if the surface contains highly spherical or hyperbolic regions, which have very different geometric structures from that of the Euclidean spaces. Stretch can also occur if a nearly flat region is unfolded in a poor manner, e.g., a square becomes a long thin rectangle. High stretch in a parameterization results in uneven sampling on the surface. On the other hand, zero stretch does not necessarily mean that we have a high-quality parameterization. Figure 1 demonstrates the problem for some parameterization technique in which every triangle is made into a patch and unfolded such that the total stretch is zero (right). However, the artificial discontinuities across the edges (the seams) become highly visible. Due to the topological constraints, it is necessary to make cuts on a closed 2-manifold in order to unfold it onto a plane. My goal is to segment a given surface into a small number of patches, each of which can be unfolded with little stretch.

Reducing stretch is related to the concept of *developable surfaces*. A surface is developable if the Gaussian curvature is zero everywhere on the surface, such as cylinders, cones, and planes. Unfolding a developable surface results in zero stretch. Let us observe that an object can be constructed by a sequence of attachments of relatively "simple shapes", such

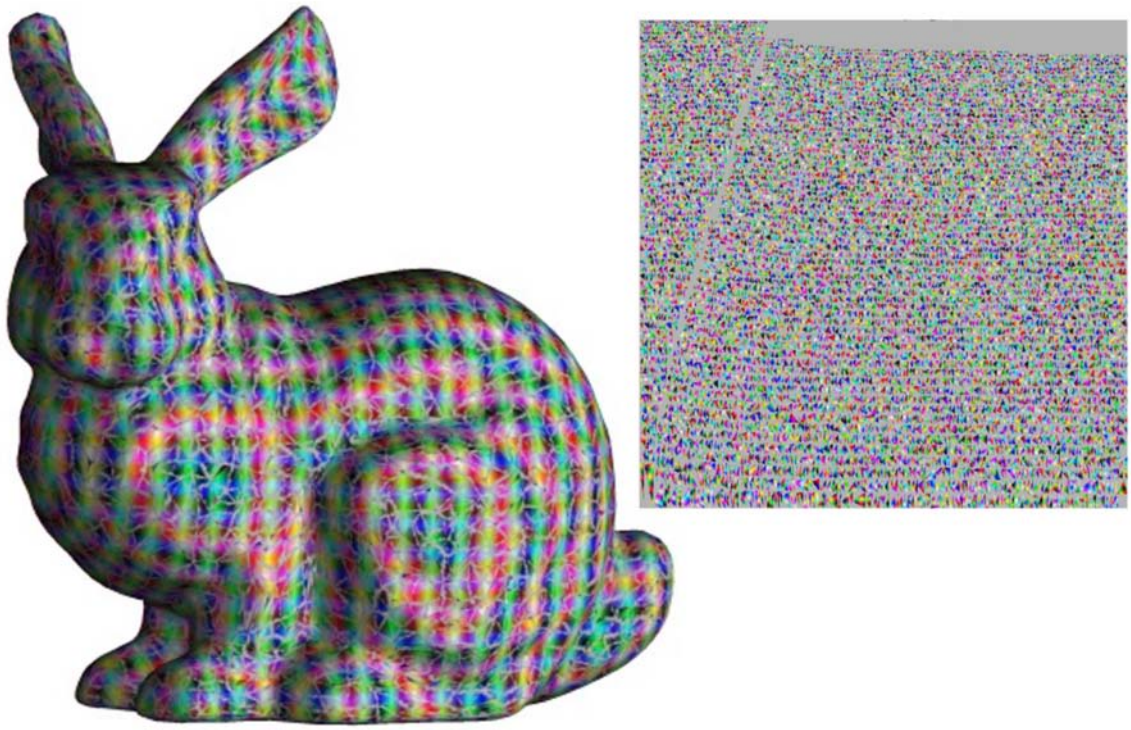

Figure 1: The zero-stretch parameterization (right) results in a large number of seams in the reconstructed texture on the bunny model (left).

as cylinders, cones, flat disks, and spheres. Reversing the attachment operations provides us with a surface decomposition with potentially little stretch. The left portion of Figure 2 illustrates such a decomposition for the bunny into four shapes: the ears, the head, and the torso. Unfolding them results in little stretch (Figure 2, right, colors are used to encode surface normal). In this thesis, I perform topological analysis using some surface-based distance functions to locate the features (handles and large protrusions) contained in a surface and to divide the surface based on these features.

Existing patch unfolding techniques are often carried out into two stages: an initial patch layout to achieve some objective such as conformal mapping, followed by an interior vertex optimization based on some geometric stretch metric. Let us observe that an ideal surface parameterization between a patch and its textural image is an isometry, that is, distance-preserving between any two points on the surface and their images in the texture space. The Green-Lagrange deformation tensor has the property that it measures anisotropic stretch faithfully and penalizes undersampling more severely than oversampling. In addition, it

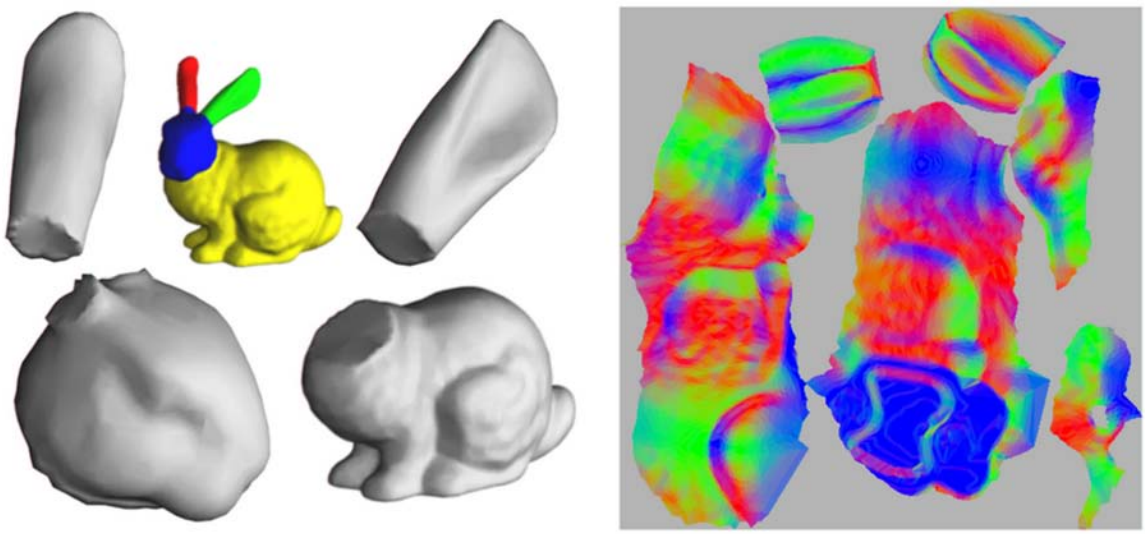

Figure 2: The feature regions (left) and the unfolded patches (right, colors are used to encode the surface normal) of the bunny based on my algorithm.

can be seen as a balance between area-preserving mappings and conformal mappings. I use this metric to guide the vertex optimization process for patch unfolding. In addition, I use *scaffold triangles* to convert the original boundary vertices into "interior" vertices, which can then be freely moved around within the same optimization framework. This is a new way of creating non-convex patches that may even have holes.

1.1.2 My Contributions

In this thesis, I present an automatic surface parameterization method that improves upon existing techniques in several ways. For patch creation, I make use of the average geodesic distance function to extract and to segment protrusions contained in the model. This results in a small number of large patches that can be unfolded with relatively little stretch. For patch unfolding, I use the Green-Lagrange tensor to measure stretch and to guide the stretch optimization process. In addition, I create a "virtual boundary" to allow the patch boundaries to be optimized without the need for checking for global self-intersections. These two techniques help produce efficient and robust unfolding that has less stretch than prior techniques. Finally, I describe an image-based quality metric for surface parameterization that implicitly takes into account stretch, seams, packing efficiency, smoothness, and surface visibility. To my knowledge, this is the first image-based quality measure for

parameterization techniques.

1.2 Vector Field Design on Surfaces

1.2.1 The Problem

Many graphics applications require an input vector field to achieve certain visual effects. For instance, example-based texture synthesis makes use of a vector field to define local texture orientation and scale. In non-photorealistic rendering, vector fields are used to guide the orientation of brush strokes and hatches. In fluid simulation, external force is a vector field that need not correspond to any physical phenomenon and can exist on synthetic 3D models. A vector field design system enables these applications to create many different visual effects by merely using different input vector fields. A vector field design system can also be used to test existing vector field visualization techniques [94, 95].

Vector field design refers to creating a continuous vector field on an input surface based on a user's specifications or application-dependent requirements. There are several challenges to the problem. First, the system should allow the user to create a wide variety of vector fields with relatively little effort. Second, the user should be able to control vector field topology, such as the number and location of the singularities in the vector field. As pointed out in [70, 39], this is necessary for applications such as example-based texture synthesis and non-photorealistic rendering, in which unwanted singularities cause artifacts in the visual appearance. Third, the system should be as interactive as possible. Finally, to create a vector field design system for surfaces defined using meshes, we will need a definition for vector field continuity on a mesh. Figure 3 gives some example vector fields defined on 3D models. These vector fields were created using the system that I describe in this thesis. Figure 4 highlights the need for controlling vector field topology in texture synthesis. Notice that the singularities at the center of the bunny's tail (left) and inside the feline's wing (right) cause the break-up of continuity of the synthesized patterns.

To achieve these goals, I use a three-stage pipeline for vector field design. First, the user quickly creates a complex vector field without being concerned about its topology. Next, the system analyzes the vector field and provides visual feedback to the user. The user

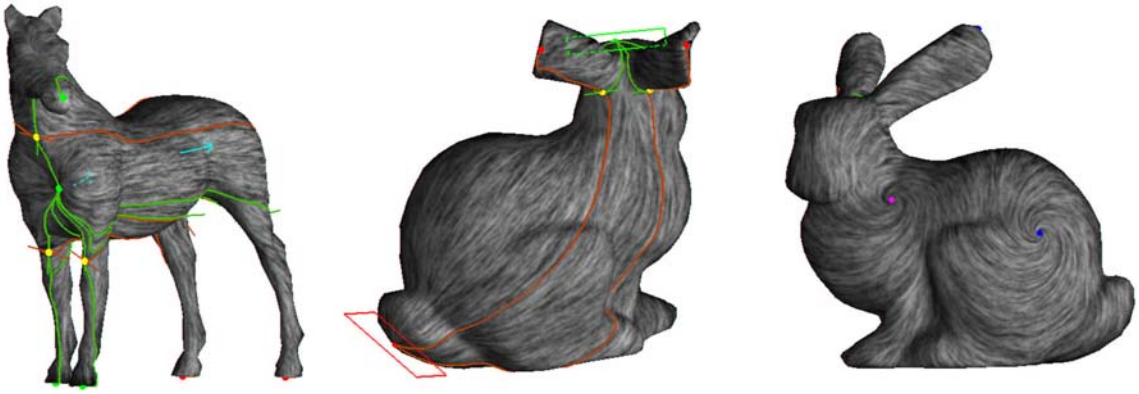

Figure 3: This figure shows three continuous vector fields defined on the horse and the bunny using my design system. Colored dots indicate the singularities in the vector fields. Also shown is the connectivity between the singularities.

can then make controlled editing operations to the current vector field, such as moving a singularity or canceling a pair of singularities. This process of iterative analysis and editing is repeated until the user is satisfied with the result.

1.2.2 My Contributions

In this thesis I present an interactive vector field design system for mesh surfaces. The system has the following attributes:

- It allows the user to create generic vector fields, not just a subclass of vector fields such as divergence-free or curl-free vector fields.
- It allows the user to generate complex vector fields with a relatively small amount of input.
- The user has control over vector field topology, including the number and location of the singularities in the vector field.
- The system works for both planar domains and curved surfaces, which are represented as meshes.

To the best of my knowledge, this is the first system that possesses all of these characteristics. To give the user control over vector field topology, I borrow results from Conley index

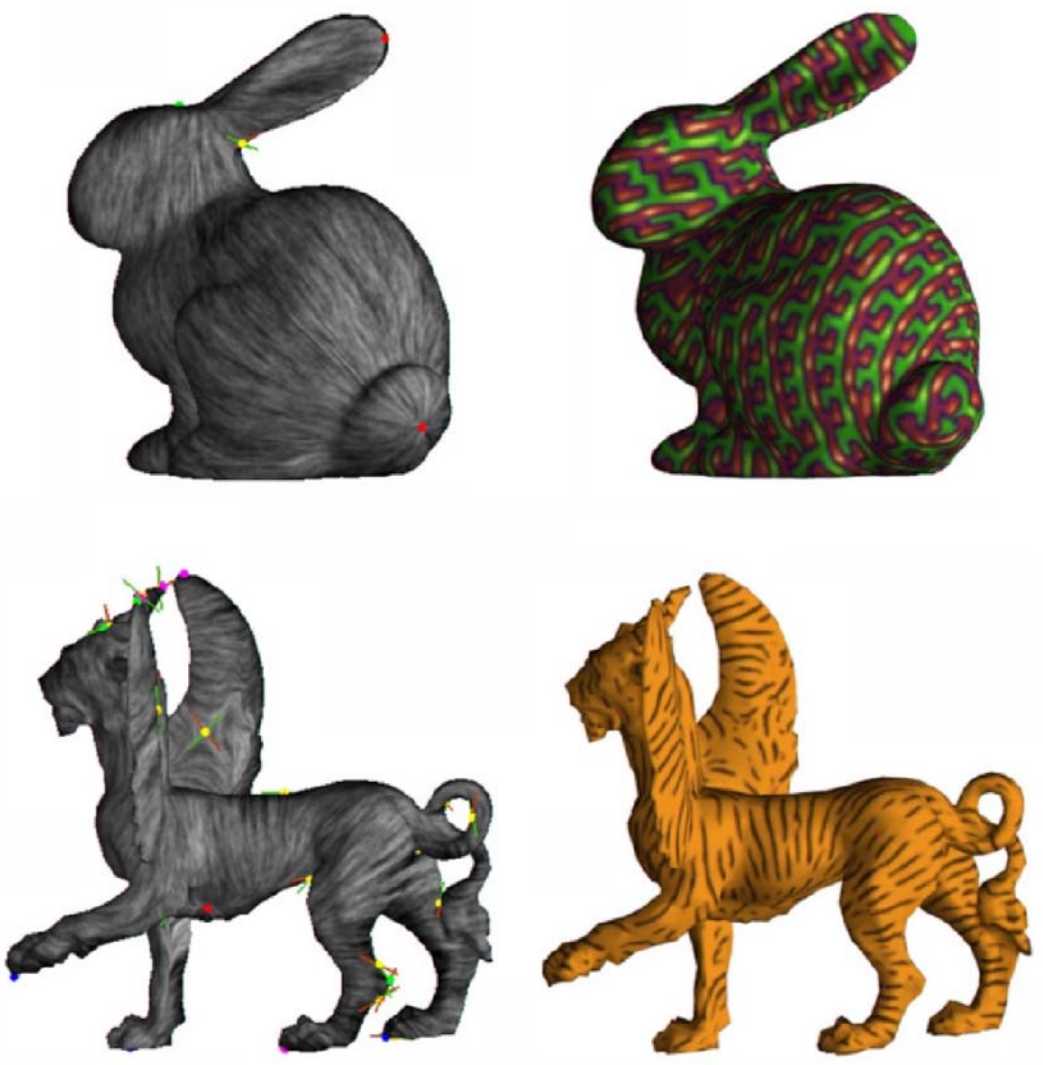

Figure 4: In texture synthesis, the singularities in the input vector fields often cause visual artifacts. Notice that the singularities at the center of the bunny's tail (left) and inside the feline's wing (right) cause the break-up of continuity of the synthesized patterns.

theory to perform topological operations such as singularity pair cancelation and singularity movement. Conley index theory is well-known in the Dynamical Systems community. However, as far as I know, this is the first time that it has been used for Computer Graphics applications. I also present a new vector field representation for mesh surfaces that is based on the geodesic polar map and parallel transport (Section 10.3.1). This representation guarantees vector field continuity. Furthermore, I introduce a new method of creating basis vector fields on surfaces (Section 10.2). To overcome the numerical difficulties caused by regions of high curl and by saddles, I use flow rotations and reflections to change any

first-order singularity into a source. I demonstrate the usefulness of this design approach with three specific applications: painterly rendering, texture synthesis, and pencil sketch illustrations of surfaces.

1.3 Thesis Organization

The remainder of the thesis is organized as follows.

Surface Parameterization: In Chapter 2, I review existing surface parameterization techniques. Then I describe my feature-based patch creation method in Chapter 3 and my new balanced stretch metric and boundary vertex optimization technique in Chapter 4. In Chapter 5, I will show the results of applying these techniques to various 3D models, and I also introduce an image-based quality metric for parameterization techniques.

Vector Field Design on Surfaces: In Chapter 6 I will review known vector field design systems and techniques. Then I review the relevant mathematical background about vector field in Chapter 7. Next I present my three-stage vector field design system for planar domains in Chapter 8 and 9, and its adaptation to surface vector fields in Chapter 10. In Chapter 11, I demonstrate the usefulness of the vector field design system with several graphics applications.

Conclusion and Future Work: In Chapter 12, I summarize my contributions for both surface parameterization and vector field design on surfaces, and discuss possible future work in these areas.

CHAPTER II

PREVIOUS WORK IN SURFACE PARAMETERIZATION

There has been a considerable amount of recent work in the graphics community on building surface parameterizations by unfolding a polygonal surface into planar patches. Much of the motivation for this is for *texture mapping*, the mapping of pixels from a rectangular domain (the *texture map*) onto a surface that is described by a collection of polygons. The parameterization problem is to subdivide the given surface into a (hopefully small) number of patches that are then flattened onto the plane and arranged in the texture map. I will first survey the uses of surface parameterizations and then review existing parameterization techniques.

2.1 Uses of Surface Parameterizations

Surface parameterization allows efficient storage of surface properties such as colors and normal, and enables interactive and high-quality rendering.

Hanrahan and Haeberli [34] use such a parameterization to allow a user to perform painting over a 3D surface with a mouse. The results of user editing operations, such as the changes in color and geometry, are recorded in a texture map through a pre-defined surface parameterization.

Texture synthesis refers to creating patterns on surfaces. Solid texture [65, 66] is an efficient texture synthesis technique for creating complex and visually pleasing patterns over an object. To create textures on a 3D surface using solid texture, one needs to evaluate a set of functions defined in $\mathbb{R}^3$ for a large number of sample points over the surface. This process is computationally expensive, and the results may be saved in a texture map. Carr and Hart [10] present a system which allows interactive design and render of solid textures with the assistance of graphics hardware. With a pre-defined surface parameterization, they rasterize the solid texture coordinates as colors in the texture map, and perform a per-pixel

texture procedure to replace the solid texture coordinates with the corresponding texture results. Recently, there have been a class of methods that perform texture synthesis based on an input image [93, 97]. These methods determine the color for a sample point on the surface by using the colors of its neighborhood to look for the best match in the input image. These example-based texture synthesis techniques often have better control over the final texture results than using solid textures, but at a higher computational cost. Therefore, the results are often saved in a texture map through a pre-defined parameterization.

Similarly, results from light simulation can be saved in a texture map, such as light maps, shadow maps, and environment maps. Parameterization can also be used to save other types of surface signals, such as surface normal [78]. This allows displacement maps and bumps maps to be created.

Parameterizations are also useful for appearance-preserving simplification. Cignoni et al. [12] create a texture map for a lower-resolution surface by sampling the texture map for the corresponding higher-resolution model. Such technique preserves the appearance of the textured map while using a much-simplified geometry. Sander et al. [78] go further in this direction by building a consistent parameterization for every mesh in the Progressive Mesh representation [42].

In addition to creating texture maps, parameterization has been used for several other graphics applications, namely, remeshing, compression, morphing, and fluid simulation.

Remeshing refers to creating a new mesh from the original surface. Remeshing can be used to improve the triangle shapes, to vary the triangle sizes according to curvature details, and to induce semi-regular tessellations. Alliez et al. [2, 1] perform efficient remeshing by using a pre-defined parameterization to compute and store surface properties, such as surface area and the curvature tensor. Furthermore, they use the curvature feature lines to determine the locations of the new vertices in the parameterization domain, therefore avoiding the numerical difficulties associated with tracing trajectories on a surface.

Gu et al. [30] build surface parameterization as a compression technique, which they call *geometry images*. Basically, the surface is cut open into one piece and flattened onto a rectangle in a plane. By recoding the locations of the seams (the places where the

surface is cut), they can reconstruct the surface. Such an image-based representation allows image-compression techniques to be applied to surface compression. To overcome the high distortion induced by the one-patch geometry images, Sander et al. [79] introduce multi-chart geometry images which allow the surface to be divided into a number of patches.

Surface morphing refers to producing a smooth transition between two 3D surfaces. The major issue in morphing is to obtain correspondences between the shapes. Lee et al. [51] obtain this correspondence between two surfaces of the same topology by first parameterizing them over some simpler domains using MAPS [50]. They construct correspondence between the simpler domains through harmonic mapping, which induces a correspondence between the original surfaces. Praun et al. [73] parameterize a set of genus-zero surfaces over a common domain to establish correspondences between the shapes. The consistent mesh parameterization constructed in this manner allows morphing between the shapes as well as shape blending and texture transfer.

In fluid simulation over 3D surfaces, Stam [87] makes use of parameterization to efficiently compute the metric tensor and to trace particles over a surface. The metric tensor is necessary to compensate the distortion introduced by the parameterization. To handle seams, he employs overlapping patches and uses transfer functions so that particle tracing can be handled across patch boundaries.

In scientific visualization and medical imaging, surface parameterizations have been used for creating an atlas. An atlas is an overview such that every part of a surface is visible simultaneously. To minimize the confusion caused by the stretch in the parameterization, Angenent et al. [3] use conformal mapping. While conformal mapping preserve angular information, it distorts distances over the surface. Saroul et al. [80] alleviate this problem by providing an interactive parameterization technique which minimizes distance distortion near the reference point and along a given direction.

Recently, octrees have been used to store colors in 3D for surface texturing without any parameterization [7, 14]. Although octree techniques are supported with programmable GPU's, they are not yet directly supported by graphics hardware.

2.2 Patch Creation

Surface parameterization is carried out in three stages:

1. *Patch creation*: a surface is divided into a number of patches, each of which is topologically a disk, possibly with holes.

2. *Patch unfolding*: a patch is flattened onto a planar domain without any foldovers.

3. *Patch packing*: the unfolded patches are packed into a rectangular map without any overlaps.

I will review existing techniques for patch creation in this section, and for patch unfolding in Section 2.3.

There are two common approaches to the patch creation problem. The first of these is to find a single cut for the surface that makes the modified surface topologically equivalent to a disk. The second major approach is to divide the surface into a collection of patches that can be unfolded with little stretch.

2.2.1 One-Patch Methods

The problem of cutting a closed and orientable surface with g handles into a single topological disk has been well researched in the Computational Topology community. Several algorithms [96, 16, 49] have been proposed to compute a *canonical polygonal schema*, which is a minimal solution to the problem in a purely topological sense. A canonical polygonal schema consists of $2g$ *non-separating cycles* [21] that pass through a single point, and these cycles form a basis of the first homological group of the surface.

For an orientable surface with g handles and n boundaries, Vegter and Yap [96] propose an algorithm to compute a canonical polygonal schema in $O(gn)$ running time and storage. Later, Lazarus et al. [49] provide two simplified algorithms with the same running time. Dey and Schiper [16] compute a polygonal schema for an orientable surface of genus g without boundary. This is achieved by first constructing a cut graph through depth-first search, then shrinking an arbitrary spanning tree in the cut graph to a point. Their algorithm runs in $O(n)$ time.

For surface parameterization, a canonical polygonal schema is not suitable since it does not take into account surface geometry. Furthermore, a canonical schema requires that all the loops to share a common vertex, which often results in unnecessary overlapping edges in the schema. Erickson and Har-Peled [21] define *cut graphs* by removing the common-vertex requirement. They also prove that the problem of finding a minimal cut graph for an orientable 2-manifold is NP-hard. Furthermore, they propose a greedy algorithm that outputs a $O(\log^2 g)$-approximation of the minimal cut graph in $O(g^2 n \log n)$ time.

Gu et al. [30] and Sheffer and Hart [84] both extend the technique of Erickson and Har-Peled [21] by making additional cuts along curves that connect a canonical polygonal schema to the "extremal" points in the surface. Gu et al. [30] measure the extremity by iteratively unfolding the surface with respect to the current cut and looking for a point with the highest geometric stretch as defined in [78]. Sheffer and Hart [84] measure the extremity using the local Gaussian curvature. In addition, they construct additional cuts so that they are short and pass through low-visibility regions contained in the surface. This technique helps reduce the visual artifacts caused by the seams.

In addition to these automatic techniques, Piponi and Borshukov [67] let the user manually cut a genus zero surface into a single patch.

Ni et al. [63] allow the user to design fair Morse functions over a surface. In their method, the user specifies desired critical points, such as local maxima and minima. Then the system automatically creates a high-quality Morse function with the least number of additional critical points. Such a Morse function induces cuts along the surface that turns the surface into a topological disk.

The one-patch approach has the virtue of creating as few seams as possible, but will often introduce large stretch between the patch and the surface. Such stretching is undesirable because different portions of the surface are represented using quite different amounts of color detail, as measured in pixel resolution in the texture map.

2.2.2 Multi-Patch Methods

The second major patch creation approach divides the surface into multiple patches. The existing techniques using this approach can be further divided into two categories: *bottom-up* or *top-down*. Bottom-up approaches create patches by merging smaller ones, while top-down approaches successively divide the surface into smaller patches. I will review them separately next.

2.2.2.1 Bottom-Up

Soucy et al. [86] propose an algorithm in which every triangle in the mesh is considered a patch. This parameterization has zero stretch at the cost of making every edge a seam. Figure 1 illustrates the visual artifacts caused by such a parameterization technique.

Later, several face-clustering algorithms have been proposed that merge patches based on some criteria that measure local *developability*. These algorithms start by labeling every triangle in the model as a patch. In each subsequent step, two patches with the least merge cost are combined as long as the resulting patch remains a topological disk. The process terminates when the minimal cost of merging any pair of remaining candidates exceeds a user-defined threshold. Similar to edge collapse for mesh simplification, the order of patch merging directly affects the quality of the final parameterization.

Malliot et al. [57] measure the merge cost using the difference between patch normal. This is based on the assumption that if a patch is approximately planar, then the face normal of all the triangles in the patch should concentrate near a single point inside the Gauss sphere. Accordingly, if two nearly planar patches have similar normal, then merging them will result in a nearly planar patch.

Garland et al. [26] measure the merge cost based on a combination of planarity and compactness. For a simply connected planar patch, compactness is the ratio between the area of the patch and the square of its perimeter. Circle achieves maximal compactness, while a largely elongated ellipse has a relatively small compactness. For parameterization, near circular patches are preferred over long thin patches since circular patches usually result in better packing efficiency. In their work, Garland et al. measure both planarity and

compactness using quadric measures, which makes the computation very efficient. Similar measures are also used by Sander et al. [78].

Another popular bottom-up approach makes use of mesh simplification, such as MAPS by Lee et al. [50]. In this approach, every face in the simplified mesh corresponds to a patch, i.e., a collection of connected triangles in the original mesh that form a topological disk. Lee et al. perform mesh simplification based on vertex removal, followed by a retriangulation (details of this type of mesh simplification can be found in [17]). In their algorithm, the choice of vertex removal is based on a combination of curvature and area information.

Guskov et al. [31] enhance MAPS by performing optimization on the patch boundaries. This leads to patches with short and smooth boundary curves, and they in turn result in high-quality remeshing (Normal Meshes).

The bottom-up patch creation techniques tend to produce a large number of small patches, and the shapes of patch boundaries often do not conform to the natural features in the surface. This is not surprising since these methods rely on local measurements of developability. While they cause relatively little stretch, the number of seams is still high.

2.2.2.2 Top-Down

Eck et al. [18] present an algorithm in which the patches are calculated as the Voronoi regions corresponding to a collection of seed points on the surface.

Lévy et al. [53] extend this technique with two enhancements. First, the seeds are selected using curvature information. For a smooth surface, this method helps identify the extremal points contained in the surface. Second, the region growing method also takes curvature into account, which allows patch boundaries to be aligned with sharp edges.

Li et al. [54] present a surface segmentation technique that is based on space sweeping according to a pre-computed *sweeping path*. They define topological and geometric functions and keep track of the critical points of these functions during the sweep. The part of the model between two consecutive critical points is considered a patch. They demonstrate the effectiveness of this approach by applying it to several test models.

Brostow et al. [9] describe a novel skeletal representation for articulated creatures captured from video. For each time frame, a surface representation is obtained for the creature. Then, they compute a single skeleton of the surface, which most likely contain large protrusions that correspond to the limbs and the head of the creature. They first locate extremal points, such as the tips of the legs and the head, using surface distance. Then, they build a skeleton with respect to each extremal point by performing region growing from the extremal point in the increasing order of surface distance. Each skeleton best represents the surface along the protrusion. For instance, the skeleton corresponding to the tip of a leg is most accurate along the leg. A high-quality skeleton for the whole model is then obtained by merging individual skeletons.

The techniques by Li et al. [54] and Brostow et al. [9] can be used to decompose a surface into feature patches. However, these patches are not necessarily topological disks. For texture mapping purposes, additional processing steps are required.

Katz and Tal [45] describe a hierarchical mesh decomposition approach using fuzzy clustering and cuts. First, they define a distance function between any two faces contained in surface based on geodesic distances between the faces and angles between the surface normal. Then, they compute the centers of the patches by finding the minimum of some average distance functional. Based on these centers, they define a probability function for every face in the model that is assigned to a patch corresponding to a center. They then use graph cuts to find the exact patch boundaries. Their method is hierarchical in the sense that a patch can be further divided. Furthermore, their technique allows a k-way decomposition in a single step and the number of patches k is automatically determined.

Sorkine et al. [85] combine patch creation and patch unfolding into a single step. In their approach, a patch is created by growing outward from a seed triangle. When a triangle is added to the patch, it is assigned a parameterization immediately. If at some point adding any triangle will cause the total stretch to be higher than a user-defined threshold, the current patch is declared finished and a new seed is selected in the unvisited part of the surface. This starts the generation of another patch. Eventually every triangle in the surface is assigned to a patch.

The top-down approaches seek to divide the surface into a (hopefully) small number of patches while having an acceptable level of stretch. This can be seen as seeking a balance between minimizing stretch and reducing the amount of seams.

The technique that I present in this thesis is a multi-patch method. Similar to other top-down approaches, I first locate and measure the tips of large protrusions (extremities). This is achieved by using a global surface distance-based function. To determine the patch boundaries, I use region growing (often used by bottom-up approaches) at various stages. As will be shown in Chapter 3, this approach is very effective in creating a small number of large patches, even for surfaces with many small geometric features.

2.3 Patch Unfolding

There have been many patch unfolding techniques described in the literature. These techniques differ in how they measure stretch and in how they unfold a patch to achieve minimal stretch.

The ideal unfolding of a patch is an isometry between the patch and its textural image. An isometry is both angle-preserving (conformal) and area-preserving. However, isometry is in general impossible to achieve for a curved patch. Several previous techniques focus on achieving conformal mappingss while others seek to find a balance between conformal mappings and area-preserving mappings.

The classical approach treats the patch unfolding problem as finding the minimum of some functional that measures the difference between a particular parameterization and an isometry. Eck et al. [18] achieves unfolding by first mapping the boundary vertices of a patch to a set of ordered points on a circle. Then the parameterization for the interior vertices are determined by solving Laplacian equations over the components of the texture coordinates. This method does not guarantee valid embedding due to the possible negative weights, unless the mesh does not contain any obtuse triangles. The underlying stretch metric is based on the Dirichlet energy.

Floater [22] devises a similar scheme to overcome the embedding problem. He observes that if the boundary set in the texture space is convex, and if the weights are non-negative,

then valid embedding is guaranteed. To achieve this, he uses weights that are inversely proportional to the edge lengths, thus positive. Later, Floater [23] proposes another scheme of non-negative weights which also takes into account of angle information in the triangulation.

These algorithms often introduce distortions because the texture coordinates for the patch boundary vertices are randomly assigned. To overcome this problem, some researchers have proposed to use different energy functions. Hormann and Greiner [43] use a variant of the Dirichlet energy to build MIPS (Most Isometric ParameterizationS). This energy formulation only requires that two boundary vertices be assigned texture coordinates.

Lévy et al. [52] uses a least-square conformal mapping formulation, which also only requires initial texture coordinates for two boundary vertices. However, due to the negative weights, this technique does not guarantee valid embedding. Desbrun et al. [15] present an intrinsic parameterization technique which is proven equivalent to the least square conformal mapping.

Lee et al. [52] overcome the initial assignment problem by adding layers of "virtual boundaries" as part of the edge springs to allow the patch boundaries to have a natural shape. However, for a complex patch, many layers are often required and this results in longer processing time.

Another class of unfolding methods do not require solving a linear system. Maillot et al. [57] unfold a patch by laying out one triangle at a time. By carefully choosing the texture coordinates for the new vertex in the triangle, stretch is minimized. In this work, stretch is measured using the Green-Lagrange deformation tensor.

Sorkine et al. [85] use a similar approach. However, they have devised a different stretch metric based on the ratio between the maximal and minimal eigenvalues of the stretch tensor. Furthermore, in their method, patch creation and unfolding are carried out in a single step.

Recently, Sheffer and deSturler [82, 83] propose to use an *angle based flattening* approach for patch unfolding. This approach measures stretch in terms of the angle deficits between the triangles on the surface and their textural images, and it removes the need for checking for global self-intersections.

Khodakovsky et al. [46] have presented another stretch metric based on the maximal and minimal eigenvalues of the stretch tensor. They obtain unfolding by solving a linear system followed by relaxation.

Sander et al. [78] propose a non-linear optimization after an initial parameterization. Basically, an interior vertex can move within its kernel to reduce stretch. They measure stretch based on the average and maximal stretch in all directions of a triangle. This optimization step can serve as a post-processing step for any parameterization technique. Later, they improve upon this technique by allowing boundary vertices to be optimized while checking for global self-intersections [77].

As I describe later, Sander's patch optimization approach [78] was an inspiration for my own work.

CHAPTER III

FEATURE-BASED PATCH CREATION

In the first stage of the parameterization pipeline, an input 3D mesh surface is segmented into a number of disjoint patches. Every patch is topologically a disk, and their union covers the surface. My goal is to segment the surface into a small number of simple shapes according to the large protrusions contained in the model. For instance, the bunny can be decomposed into the ears, the head, and the body. Every simple shape can then be approximated by an ellipsoid, for which I create one or two patches and unfold them with relatively little stretch. There are a number of issues that must be addressed. First, how do we locate protrusions and measure their sizes? A bump is a small protrusion, whose presence in a patch in general does not increase the stretch dramatically. However, segmenting it will cause more seams. Second, how do we divide the surface into simple shapes according to protrusions? Finally, for a high-genus surface, we need to address handles. Locally, a handle in the surface often looks like a protrusion, yet they have very different topological behaviors. A single closed curve can be used to separate a protrusion from the rest of the surface. For a handle, at least two such curves are required.

To address these issues, I describe a feature-based method in which patch creation is carried out in three stages:

1. *genus reduction*: a surface with handles (non-zero genus) is converted into a genus zero surface.
2. *feature identification*: a genus zero surface is divided into a number of relative simple shapes.
3. *patch creation*: every simple shape from feature identification is cut into one or two topological disks.

For both genus reduction and feature identification, I build a surface-based Reeb graph

based on the *average geodesic distance* function introduced by Hilaga et al. [40]. This graph consists of vertices and edges of the mesh surface, and I call it an *embedded Reeb graph*. When properly constructed, the embedded Reeb graph Γ reveals the locations of the handles and protrusions in the surface. Since my goal is to create patches that are topological disks, I need to perform these operations in a topologically consistent manner. For this purpose, I use surface region growing for all three stages: genus reduction, feature identification and patch creation. By starting from an initial triangle, I grow a region by adding one triangle at a time until the whole surface has been covered or when some other stopping criterion has been met. Next, I will describe the average geodesic distance function and the embedded Reeb graph that it induces.

3.1 AGD*: the Average Geodesic Distance Function*

The *average geodesic distance* function was introduced by Hilaga et al. [40] for the purpose of shape matching. This is a function $A(\mathbf{p})$ that takes on a scalar value at each point $\mathbf{p}$ on the surface S. Let $g(\mathbf{p}, \mathbf{q})$ be the geodesic distance between two points $\mathbf{p}$ and $\mathbf{q}$ on the surface. Then the average geodesic distance is defined as follows:

$$A(\mathbf{p}) = \frac{\int_{\mathbf{q}\in S} g(\mathbf{p}, \mathbf{q}) d\mathbf{q}}{Area(S)} \tag{1}$$

$A(\mathbf{p})$ is a member of the following set of functions with $n = 1$.

$$A_n(\mathbf{p}) = \sqrt[n]{\frac{\int_{\mathbf{q}\in S} g^n(\mathbf{p}, \mathbf{q}) d\mathbf{q}}{Area(S)}} \tag{2}$$

When $n \to \infty$, $A_\infty(\mathbf{p}) = \lim_{n\to\infty} A_n(\mathbf{p}) = \max_{\mathbf{q}\in S} g(\mathbf{p}, \mathbf{q})$, which measures the maximal distance from any point $\mathbf{q}$ on the surface to $\mathbf{p}$. Let $\mathbf{p}_\infty$ be the global minimum of A_∞.

Theorem 3.1.1. *$\forall \mathbf{q} \in S$, $A_\infty(\mathbf{q}) \leq 2A_\infty(\mathbf{p}_\infty)$.*

Proof. Let $\mathbf{r}$ be such that $g(\mathbf{q}, \mathbf{r}) = \max_{\mathbf{u}\in S} g(\mathbf{q}, \mathbf{u})$. Since g is a metric, it satisfies the triangle inequality. Therefore,

$$g(\mathbf{q}, \mathbf{r}) \leq g(\mathbf{p}_\infty, \mathbf{q}) + g(\mathbf{p}_\infty, \mathbf{r}) \leq 2 \max_{\mathbf{u}\in S} g(\mathbf{p}_\infty, \mathbf{u}) \tag{3}$$

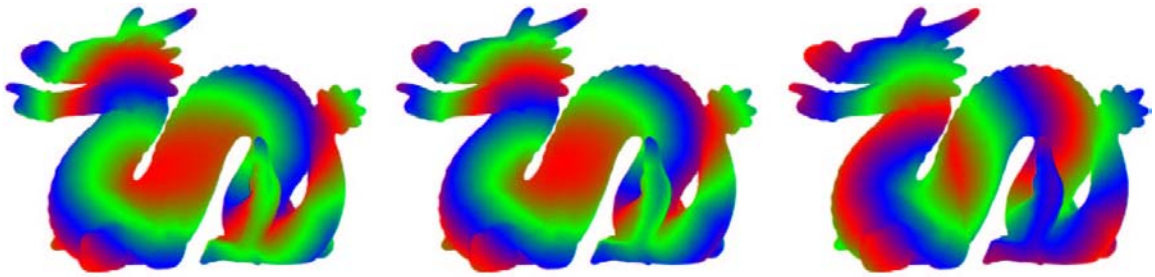

Figure 5: This figure compares three AGD functions on a dragon model (Section 3.1). From left to right are: AGD_1, AGD_2, and AGD_∞. Notice that AGD_2 is smoother than both AGD_1 and AGD_∞.

which is equivalent to the statement of the theorem. □

Let us define for $n = 1, 2, ..., \infty$ a normalized version of $A_n(\mathbf{p})$,

$$AGD_n(\mathbf{p}) = \frac{A_n(\mathbf{p})}{\min_{\mathbf{q}\in S} A_n(\mathbf{q})} \tag{4}$$

For any $n \geq 1$, AGD_n has several useful properties. First, its value measures how "isolated" a point is from the rest of the surface. A global minimum $\mathbf{p}$ of AGD would be a point that is closest to everything. Second, its local maxima coincide with the tips of the geometric features contained in the model. Third, it is scale-invariant and can be used to compare features from different shapes. Figure 6 (left) shows a polygonal model of a dinosaur, color-coded according to AGD_2. The red region on the dinosaur's belly signifies that the points in this region have low values of AGD_2. Higher values adjacent to this middle region are colored in green and then in blue. The colors then cycle repeatedly through red, green, and blue. Note that the tips of the large features of this object (legs, horns, tail) are marked by local maxima of AGD_2. (In subsequent sections I will use the term *local maxima of AGD* and *tips* interchangeably.) In practice, I use AGD_2 since it seems to produce smooth results. Figure 5 compares the level sets of the following three functions on the dragon: AGD_1 (left), AGD_2 (middle), and AGD_∞ (right). From now on, I will use the term AGD to mean AGD_2.

For a genus zero surface, I use AGD to identify and to measure its geometric features. The tip of a protrusion is a local maximum. Larger values at the local maxima signify larger protrusions. In practice, I consider any local maximum $\mathbf{p}$ to be the tip of a geometric

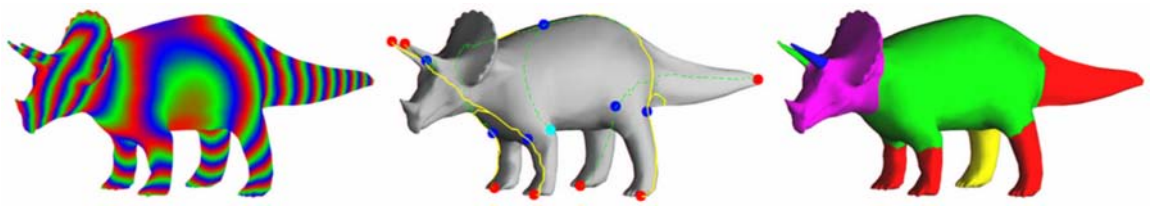

Figure 6: The *average geodesic distance* (AGD) of the dinosaur model is color-coded in the left portion of this figure. The global minimum is underneath the belly, in red. Level sets are painted in repeated patterns of red/green/blue. Notice that the level sets change topology and end on the tips of features. The middle portion shows the *embedded Reeb graph* created by surface growing based on the AGD. The local maxima are shown with red spheres, and saddle points are highlighted by the blue spheres. Successive *critical points* are connected by surface paths shown in solid yellow (visible) and dash green (hidden). The right portion shows the final surface segmentation into feature regions that were identified based on the embedded Reeb graph.

feature if $AGD(\mathbf{p})$ is larger than 1.4. In fact, I find that choosing any number in $[1.3, 1.5]$ as the threshold for minimal feature size produces reasonable results.

Computing AGD exactly would be quite costly. By following closely the algorithm by Hiliga et al. [40], I quickly compute a satisfactory approximation of AGD. Briefly, the geodesic distances are not calculated from all points $\mathbf{q}$, but rather from a small number of evenly spaced points on the surface S. To obtain the point set, I apply Lloyd's algorithm for generating centroidal Voronoi diagrams on meshes [64]. Initially, the points are placed randomly on the surface and in different triangles. Next, the Voronoi regions for these points are computed, and every point is moved to the center of its Voronoi region. Repeating this process eventually results in an evenly spaced point set. In practice, I use 20 points for all the test models, and repeat the Voronoi centering process five times because of its quick convergence. I find the geodesic distances from each of these points to all other points efficiently using the fast marching method for surfaces of Kimmel and Sethian [47].

3.2 Embedded Reeb Graph

To find the handles and large protrusions in a model, I perform topological analysis of AGD and construct an embedded Reeb graph Γ. The leaf nodes of Γ are situated at the tips of protrusions of the surface and the loops in Γ reveal the existence of handles. Γ is constructed by performing region growing in the increasing order of AGD and by tracking

the topological changes in the wavefront. This is based upon ideas from *Morse Theory* [60] and the *Reeb graph* [74], which I will review here.

Let f be a smooth function defined on a smooth surface $S \subset \mathbb{R}^3$. For any point $\mathbf{p} \in S$, let $\mu = (u, v)$ be a parameterization of some neighborhood of $\mathbf{p}$ over $\mathbb{R}^2$ such that $\mu(0, 0) = \mathbf{p}$. The *gradient* $\bigtriangledown f$ and the *Hessian* Hf are defined as follows:

$$\bigtriangledown f = \begin{pmatrix} \frac{\partial f}{\partial u} \\ \\ \frac{\partial f}{\partial v} \end{pmatrix}, Hf = \begin{pmatrix} \frac{\partial^2 f}{\partial u^2} & \frac{\partial^2 f}{\partial u \partial v} \\ \\ \frac{\partial^2 f}{\partial u \partial v} & \frac{\partial^2 f}{\partial v^2} \end{pmatrix} \tag{5}$$

$\mathbf{p}$ is a *critical point* of f if $\bigtriangledown f(\mathbf{p}) = 0$. Otherwise, $\mathbf{p}$ is *regular*. A critical point $\mathbf{p}$ is said to be *non-degenerate* if $Hf(\mathbf{p})$ does not have any zero eigenvalues. In this case, $\mathbf{p}$ can be classified as a minimum/saddle/maximum if $Hf(\mathbf{p})$ has zero/one/two negative eigenvalues. f is *Morse* over S if f does not have any degenerate critical points. Morse theory relates the critical points of a Morse function to the topology of the underlying surface. For instance, when S is a closed orientable 2-manifold with *Euler characteristic* $\chi(S)$ (equals twice the number of handles plus two), the following is true for any Morse function f defined on S with α maxima, β saddles, and γ minima.

$$\alpha - \beta + \gamma = \chi(S) \tag{6}$$

Banchoff extended Morse theory to triangular meshes [6]. A Morse function f induces a Reeb graph, which is a combinatorial graph that reveals the critical points of f and their connectivity. Figure 7 illustrates an example Reeb graph that corresponds to the vertical height function defined on a 3D surface. Many applications make use of Reeb graphs. Hilaga et al. [40] construct multi-resolution Reeb graphs based on the average geodesic distance function. They demonstrate that such graphs can be used for shape matching, especially for articulated models such as two poses of an animal model. Wood et al. [101] compute Reeb Graphs for locating and removing small handles in isosurfaces.

I construct an embedded Reeb graph corresponding to AGD. AGD is in general not a Morse function. For instance, it is a constant function for a sphere, and every point on the

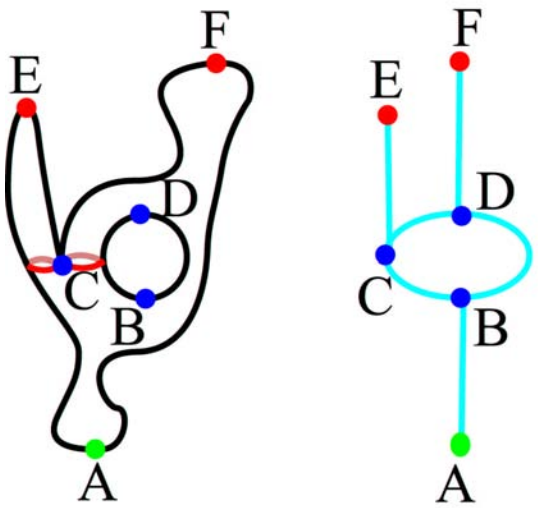

Figure 7: An example Reeb graph (right) induced by the vertical height function for a 3D surface of genus one (left). The critical points are highlighted by colored spheres (red for maxima, green for minima, and blue for saddles). The number of independent loops in the Reeb graph equals the number of handles in the surface.

sphere is a degenerate critical point. Axen and Edelsbrunner [5] show that a function can be perturbed into a Morse function with surface wave traversal, provided the mesh is properly subdivided. I use a similar strategy except that critical triangles are recorded instead of critical vertices. Hilaga et al. [40] have used a more common definition of Reeb graphs, in which Reeb graph is related to the topology of the level sets of the underlying function. This definition does not require the function to be Morse.

My algorithm for building an embedded Reeb graph Γ starts with computing AGD for every vertex. For a triangle $T = \{v_1, v_2, v_3\}$, I define

$$AGD(T) = \min\{AGD(v_1), AGD(v_2), AGD(v_3)\} \tag{7}$$

Starting with a triangle whose AGD value equals the global minimum, I add one triangle at a time in the increasing order of the AGD until the surface is covered. At any given time during region growing, the boundary of the visited region consists of a number of loops. When a triangle is added to the region, it is labeled according to one of the following five criteria:

1. Minimum: where one new boundary loop starts. For surface segmentation, there is

only one such triangle, one of the global minima of the AGD.

2. Maximum: where one boundary loop vanishes. These are the tips of the geometric features.
3. Splitting saddle: where one boundary loop intersects itself and is split into two.
4. Merging saddle: where two boundary loops intersect and merge into one. This signifies the formation of a handle.
5. Regular: where the number of boundary loops does not change.

A triangle that is not regular is a critical triangle. Let n be the genus of the surface and let N_{max}, N_{min}, N_{ss}, and N_{ms} be the number of the triangles that are maxima, minima, splitting saddles, and merging saddles. Then we have,

$$N_{ms} = n \tag{8}$$

$$N_{max} - N_{ss} + N_{ms} + N_{min} = 2 \tag{9}$$

Equation 9 corresponds to the handlebody decomposition of a closed and orientable piecewise linear 2-manifold [76]. Interested readers may refer to [56] for more details. In my algorithm, $N_{min} = 1$. Furthermore, I mark the center of a critical triangle as the position of the corresponding critical point. The region of the surface swept out between a pair of successive critical triangles (not including these critical triangles) is homeomorphic to a cylinder without caps. Next, I describe ideas used to create the embedded Reeb graph.

Let A and B be a pair of critical triangles, and assume that A is visited earlier than B. A is referred to as the *parent critical triangle* and B as the *child critical triangle*. For a genus zero surface, every child critical triangle has a single parent. For surfaces with genus greater than zero, a child critical triangle may have one or two parents. Let R_{AB} be the connecting region between A and B, which consists of a set of regular triangles $= \{T_1, ..., T_k\}$ in the order of which they are visited. There is a shortest path that connects A and B using the edges of the set of triangles $\{A, B\} \bigcup R_{AB}$. I construct the embedded Reeb graph by finding the shortest paths between every pair of parent/child critical triangles.

As I mentioned earlier, the embedded Reeb graph Γ is much like a Reeb graph that corresponds to the function AGD. It reveals the distribution of the geometric features over the surface. The middle image of Figure 6 shows the embedded Reeb graph of the dinosaur. Local maxima are highlighted with red spheres, while blue spheres indicate the location of splitting saddle points. The global minimum is marked with a light blue sphere on the belly. Successive critical points are connected by paths on the surface, which are drawn in solid yellow (visible) and dash green (hidden). Note that the local maxima coincide with the tips of the geometric features (horns, feet, and tail).

Since complex surfaces often contain many small protrusions (bumps), the embedded Reeb graph can contain an excessive number of local maxima and saddle points. This increases the subsequent processing time since the number of features is much more than what we consider large (or "persistent" as described in [20]). I use a filtering scheme to weed out extra local maxima and splitting saddle points. This is achieved by altering the order in which triangles are added during surface growing. To be more specific, let $\mathbf{t}$ be the unvisited triangle with the smallest value of AGD. If adding $\mathbf{t}$ causes a boundary to split, I look for other triangles which could be added without causing the topology of the region boundary to change. If one of these triangles, $\mathbf{t}'$ satisfies:

$$AGD(\mathbf{t}') < \alpha AGD(\mathbf{t}) \tag{10}$$

where α is a global filtering constant, then $\mathbf{t}'$ is added instead of $\mathbf{t}$. When there are multiple choices, I choose the triangle with the smallest AGD value. The filtering process is related to the concept of *topological persistence and simplification* [20], but with a different scalar function and a different measure for persistence. Also, the simplification process is implicit. In Figure 8, the filtering scheme is applied to the bunny surface with three different α's: 1.0001 (left), 1.1 (middle), 1.5 (right). Notice the excessive number of saddle points and maxima that appear in the head and the paws when $\alpha = 1.0001$ (left). When $\alpha = 1.1$ (middle), the local maxima that reveal large geometric structures are kept (the tips of ears and the center of the tail). Excessive filtering will result in a trivial embedded Reeb graph if α is too high ($\alpha = 1.5$, right). This becomes a classical trade off between de-noising and

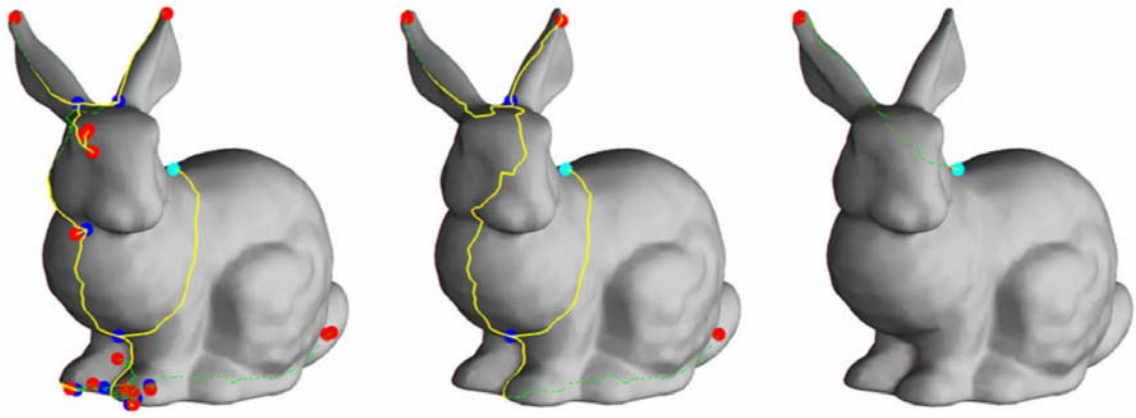

Figure 8: Embedded Reeb graphs for the bunny surface with different filtering constants α: 1.0001 (left), 1.1 (middle), and 1.5 (right). I use 1.1 as the filtering constant for all the test models.

over-blurring. In practice, I find $\alpha \in [1.1, 1.3]$ works well for all the test models, and I use $\alpha = 1.1$.

For a genus $n > 0$ surface, there are n loops in the embedded Reeb graph Γ that are homologically inequivalent and form the bases of all loops in Γ. Later in Section 3.4, I describe how to use these loops for genus reduction, that is, converting a genus n surface into a genus zero surface.

3.3 Feature Identification

Once we find the tip of a geometric feature, we need to construct a closed curve γ on the surface that separates the feature from the remaining body. Using the terminology from [21], this closed curve is referred to as a *separating cycle*. A separating cycle is constructed in two steps. First, I find a separating region R, and then I construct a separating cycle γ from R.

To find a separating region for a maximum $\mathbf{p}$ of a feature, I first calculate the function $f_{\mathbf{p}}(\mathbf{q}) = g(\mathbf{p}, \mathbf{q})$, the surface geodesic distance function with respect to the tip $\mathbf{p}$. $f_{\mathbf{p}}$ is normalized to take on values in $[0, 1]$. Let us consider the regions bounded by the iso-value curves of this function. Specifically, the interval $[0, 1]$ is divided into k equal sections, and then using region growing from $\mathbf{p}$ the surface is partitioned into bands based on the values

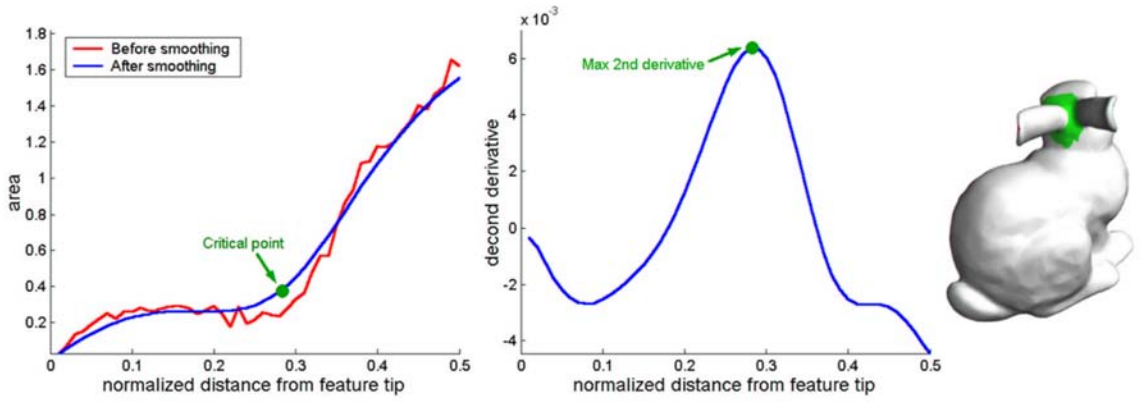

Figure 9: In this figure a separating region is constructed for one of the bunny's ears. In the left portion is the graph of $A(x)$, the area of the regions given by evenly spaced distances from the ear's tip (red), and the smoothed areas (blue). The second derivative of this function is then calculated (middle). The maximum corresponds to the place where the ear intersects with the head, shown in green in the right.

of $f_{\mathbf{p}}$ in these intervals:

$$M_i := \{\mathbf{q} \in S | \frac{i-1}{k} \leq f(\mathbf{q}) \leq \frac{i}{k}\} \tag{11}$$

$$A_i := Area(M_i) \tag{12}$$

The construction of level sets in Equation 11 is inspired by Morse theory. The area of this sequence of bands changes slowly along a protrusion, but it changes abruptly when the feature grows into the surface. I find the separating region by analyzing $\{A_i\}$, which is treated as a continuous function $A(x)$. Along a perfect cylindrical feature, $A(x)$ is constant. In case of a cone, the function grows linearly. At places where a protrusion intersects with the main body, $A(x)$ will have a sudden increase, and this will be the boundary of the feature. I find these increases by looking for maxima in the second derivative of $A(x)$. To eliminate small undulations in $A(x)$, I first low-pass filter $A(x)$ using a Gaussian function for a number of times (I use 30 for all the test models). Figure 9 illustrates this process when the point $\mathbf{p}$ is the tip of the bunny's ear.

Let m be the location where $A(m)$ takes on its maximum value. I define the separating region as a set of points $\mathbf{q} \in S$:

$$R := \{\mathbf{q} \in S \mid m - \epsilon \leq f_p(\mathbf{q}) \leq m + \epsilon\} \tag{13}$$

I typically use $\epsilon = 0.02$. I intentionally make R large for two reasons. First, poor triangulations may cause R to be non-separating, that is, not containing a closed loop which separates the tip $\mathbf{p}$ from the rest of the surface. Second, I would like more flexibility to allow the separating cycle (within this region) to be short and smooth.

The topology of a separating region R can be rather complex when other features join the surface in nearby places. The only guarantee R provides is that it indeed separates the feature from the rest of the surface. From R, a separating cycle γ is produced as follows. First, the separating region R is shrunk into its skeleton, i.e., a collection of edges on the surface that separate the feature from the rest of the surface. Dangling edges are removed as well. Gu et al. [30] performed similar operations to produce geometry images from meshes. Next, I find a separating cycle ρ from this skeleton. Finally, I construct another separating cycle γ based on ρ but is in general shorter and smoother. These operations are easy to implement on meshes, and I describe them in detail below. Figure 10 illustrates this process with a synthetic example.

1. Reduce a separating region R into its skeleton and remove dangling edges. This is achieved by treating the separating region as a 2-complex (with boundary edges) and repeatedly performing "elementary collapses"[44], in which a triangle with at least one boundary edge is removed from the complex along with one of the boundary edges. In the end, all 2-cells (triangles) are removed and the 2-complex is reduced to a 1-complex. When there are multiple choices of boundary edges for collapse, I select the edge with the largest AGD value, which tends to be closer to the feature tip $\mathbf{p}$ than the other edges in the 2-complex. The resulting graph is a skeleton of R with dangling edges. I remove the dangling edges by performing elementary collapses on the 1-complex. This results in a collection of loops, one of which meets our requirement as the separating cycle. The others fall into two categories: separating cycles for some geometric features inside the feature region, and separating cycles for some geometric features outside the feature region.

2. Eliminate separating cycles inside and outside the feature region. To remove the loops outside the feature region, I perform region growing from the feature tip $\mathbf{p}$ with the constraint that no triangle can be added that crosses an edge in the loops computed from

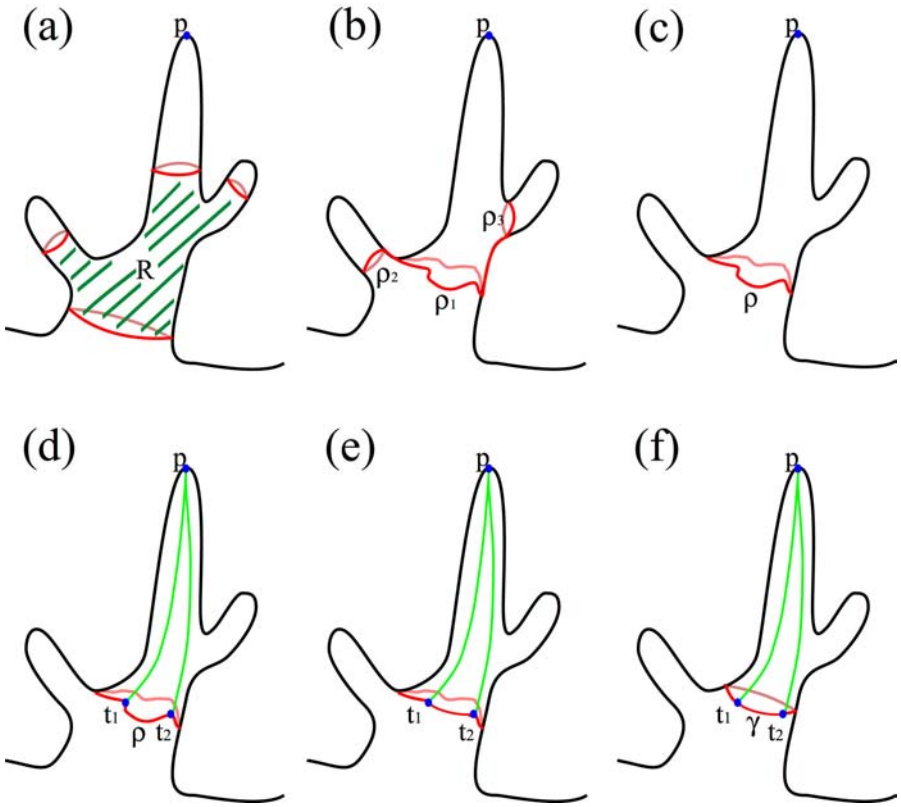

Figure 10: This figure illustrates the process in which a separating region is used to produce a separating cycle. In (a), a separating region R (shaded) for $\mathbf{p}$ is bordered by the red-curves. Next (b), through elementary collapse, R is reduced to its skeleton, which consists of three loops: ρ_1, ρ_2, and ρ_3. ρ_2 is not reachable from $\mathbf{p}$, and ρ_3 has a higher average AGD value than ρ_1. By eliminating them, I obtain a separating cycle $\rho = \rho_1$ (c). To smooth ρ, I find two points $\mathbf{t}_1$ and $\mathbf{t}_2$ in ρ and find shortest paths that connect $\mathbf{t}_1\mathbf{p}$ and $\mathbf{t}_2\mathbf{p}$ (green curves in (d)). These two curves divide the region bounded by ρ into two sub-regions. I construct a shortest path between $\mathbf{t}_1$ and $\mathbf{t}_2$ within each sub-region (red curves in (e) and (f)). The union of the two paths is a separating cycle γ that is shorter and smoother than ρ.

step 1. This way the loops outside the feature region become unreachable from $\mathbf{p}$. For loops inside the feature region, the AGD values are in general greater on their vertices than those on the separating cycle. Therefore, they can be easily identified and discarded. This step produces a separating cycle ρ.

3. Shorten and smooth the separating cycle ρ. I choose two vertices $\mathbf{t}_1$ and $\mathbf{t}_2$ on ρ that are the closest to the feature tip $\mathbf{p}$. I then find two paths that connect $\mathbf{t}_1$ and $\mathbf{t}_2$ to $\mathbf{p}$, respectively. The two paths collectively divide the feature region into two disjoint regions. Within each region there is a shortest path between $\mathbf{t}_1$ and $\mathbf{t}_2$. Together, the two shortest

paths form a separating cycle, which tends to shorter and smoother than ρ. By repeating this process twice, a shorter and smoother separating cycle γ is obtained.

Figure 10 illustrates this process. For a feature point $\mathbf{p}$ and the separating region R (a), R is reduced to its skeleton through elementary collapses (b). Next, the separating cycles that are inside and outside the feature region of $\mathbf{p}$ are eliminated (c). In the bottom row, separating cycle ρ is shortened and smoothed through step 3 (d-f).

The separating cycle γ divides S into two surfaces with boundaries. I eliminate these boundaries by "filling in" the two holes with triangles. Basically, I compute $\mathbf{c}$, the average position of the boundary vertices and make $\mathbf{c}$ a new vertex. I then triangulate the hole by connecting $\mathbf{c}$ to the vertices on the boundary. The filler triangles are subdivided twice, followed by Laplacian smoothing on the newly created vertices [88]. Some filler triangles can be seen where the head has been separated from the neck of the bunny in Figure 2. These filler triangles are flagged so that they have minimal effect on patch unfolding. They become what I call *scaffold triangles*, to be described later.

The feature identification process is repeated for the resulting surfaces until the original surface is divided into a set of features regions and there are no more feature regions to be found. Figure 2 shows the result of this process on the bunny, in which four regions were created.

Observe that the algorithm of creating a separating cycle assumes that a single loop always divides the surface into two disjoint regions. This is not true for surfaces with non-zero genus. For these surfaces, the topology of a separating region can be arbitrarily complex if many small handles are clustered together. It is rather difficult to construct a separating cycle from the separating region, let alone have it be short and smooth without causing topological inconsistency. To avoid dealing with this situation, I perform genus reduction before feature identification. Genus reduction converts a genus $n > 0$ surface into a genus zero surface.

3.4 Genus Reduction

For a genus n surface ($n > 0$), a loop on the surface may not separate the surface into two disjoint connected components. Loops with this property are associated with the elements of the first homology group, which form an Abelian group with $2n$ generators. Using the terminology from [21], these loops are *non-separating cycles*. Conceptually, the easiest way to think of how these loops arise is to imagine a hollow handle connected to the rest of the surface; one of the loops cuts across the handle and the other follows the handle. Observe that for the first type of loop there are two "passages" back to the rest of the surface. My goal for genus reduction is to identify an appropriate non-separating cycle across the handle and to use it to cut and block off the handle, thus reducing it to a protrusion. This process is repeated until a genus zero surface is generated. Erickson and Har-Peled [21] proved that the problem of finding a minimal-length cut needed to turn a surface into a disk is NP-hard, so heuristics are used in practice to find cuts that are short in length.

Genus reduction may be performed using a number of already existing techniques, including [32, 49, 21, 30, 84, 101]. Guskov and Wood [32] locate a handle in the surface by performing region growing from a seed triangle on the surface and by tracking the topological changes in the region boundaries. When two boundaries merge (a merging saddle occurs), a handle is found. Inside the active (visited) region, they produce two paths that connect the merging saddle point to the seed. Finally, they create a non-separating cycle from the paths and cut the surface along the cycle. This operation lowers the genus of the surface by one. Since the initial seed triangle is randomly selected, in the worst case the region growing process may cover a large percentage of the surface before a small handle is located. To overcome this problem, they allow the user to select a radius ϵ for region growing, and they make use of a pre-processing step to reduce the number of candidates for the initial seed triangles. The technique that I am about to describe is based on similar topological ideas. However, I choose to perform genus reduction using the embedded Reeb graph Γ and the same distance function (AGD) that I use for feature identification. This eliminates the need for a pre-defined radius for region growing, and it guarantees to find a handle each time region growing is performed.

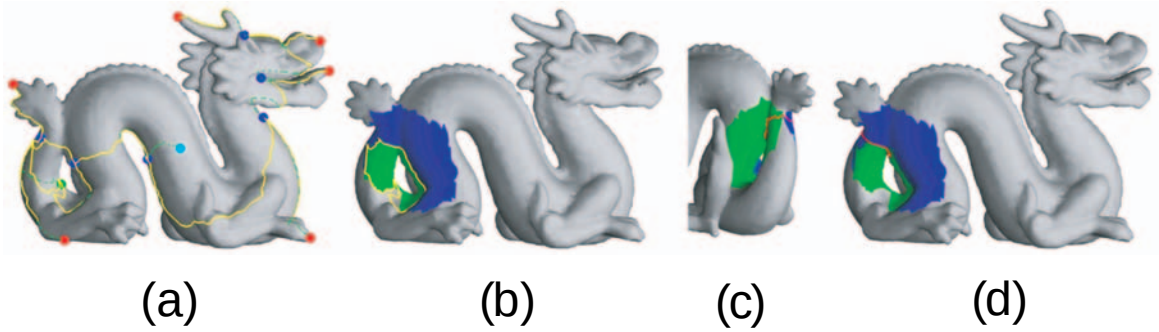

Figure 11: This figure shows how I perform genus reduction for the dragon. After the embedded Reeb graph Γ is computed ((a), the graph colored in yellow), I find the independent non-separating cycle contained Γ ((b), yellow) and perform region growing from this loop in both directions until the wavefronts meet ((b), the blue and green regions). The meeting point is shown in (c). I find a path within the green and blue regions that connect the meeting point to the original non-separating cycle. These two paths form a non-separating cycle (red in both (c) and (d)), which can be used for turn a handle into one or two protrusions.

At each step of genus reduction, I compute a non-separating cycle γ on the surface, cut the surface open along γ, and fill in the holes on both sides with scaffold triangles. Figure 11 shows the process on the dragon model, a genus one surface.

First, I compute the embedded Reeb graph Γ induced by AGD ((a), the yellow graph) and locate all the basis loops in Γ ((b), the yellow loop). Recall that a merging saddle point $\mathbf{q}_i$ signals that a handle has formed (Section 3.2) ((a), the green sphere). The start of a handle is a splitting saddle point $\mathbf{p}_i$. I trace back from the merging saddle to the corresponding splitting saddle along the paths in Γ. The two paths connecting the two saddle points is a basis loop in the embedded Reeb graph.

Next, for each basis loop ρ, the splitting saddle point $\mathbf{p}_i$ is in general located near the end of a passage that connects the handle to the rest of the surface. I create a nearby non-separating cycle γ for one of the passages by performing region growing from ρ in the increasing order of the distance function from $\mathbf{p}_i$. ρ is considered to be two oriented loops with opposite directions. Together they bound a region with two boundaries and no interiors. Since the surface has handles, the two boundaries will meet at a merging saddle point $\mathbf{r}$. Figure 11 shows the shapes of the two regions ((b), blue and green) at the time when they meet $\mathbf{r}$ (in (c)). Within each region, there is a shortest path between $\mathbf{r}$ and ρ.

Figure 12: This figure displays the non-separating cycles that are used for genus reduction for the happy Buddha (left), the dragon (upper-right), and the feline (lower-right). Each separating cycle consists of a sequence of edges in the surface that form a closed loop. Notice they appear in intuitive places. Furthermore, they tend to be short, smooth, and non-winding.

Together, the two paths form a non-separating cycle γ ((d), the red loop) that is in general shorter and smoother than ρ.

Finally, the surface is cut open along γ and the resulting holes are filled in. This reduces the genus of the surface by one. The process is repeated until the surface becomes a genus-zero surface, at which point it is ready for feature identification (Section 3.3). Figure 12 shows the non-separating cycles that are generated by my genus reduction algorithm for three genus $n(> 0)$ surfaces. Notice that these loops appear in intuitive places. Furthermore, they appear to be short, smooth, and non-winding.

3.5 Converting Regions to Patches

Through genus reduction and feature identification, a surface is decomposed into a set of simple shapes (features) that are topological spheres without large protrusions. The next task is to create one or two patches (topological disks) for every sphere so that unfolding them results in little stretch. This is carried out in two stages. First, the feature shapes

are classified as belonging to one of the three profiles: a long ellipsoid, a flat ellipsoid, and a sphere. Next, patches are created for every feature shape based on its profile.

The classification step requires computing the eigenvalues of a covariance matrix for every feature shape F, which is a triangular mesh. I compute the covariance matrix M_F by following the method of Gottschalk et al. [29] that begins by thinking of F as having been sampled "infinitely densely" by points over its surface. Suppose that the set of vertices of F is $\{V_1, V_2, ..., V_m\}$ and the coordinate for vertex V_i is v_i. Also suppose that set of triangles of F is $\{T_1, T_2, ..., T_n\}$. The vertices of a triangle T_i are $V_{T_i^0}$, $V_{T_i^1}$, and $V_{T_i^2}$. First I calculate the mean μ of these points in closed form by integrating over all the triangles.

$$\mu = \frac{1}{6n} \sum_{i=1}^{n} \frac{v_{T_i^0} + v_{T_i^1} + v_{T_i^2}}{Area(T_i)} \tag{14}$$

Similarly, I compute the covariance matrix M_F of the surface points relative to μ. Let $\overrightarrow{v_i} = v_i - \mu$. Then

$$M_F = \frac{1}{24n} \sum_{i=1}^{n} Area(T_i)(\overrightarrow{v_{T_i^0}}\overrightarrow{v_{T_i^0}}^T + \overrightarrow{v_{T_i^1}}\overrightarrow{v_{T_i^1}}^T + \overrightarrow{v_{T_i^2}}\overrightarrow{v_{T_i^2}}^T + (\overrightarrow{v_{T_i^0}} + \overrightarrow{v_{T_i^1}} + \overrightarrow{v_{T_i^2}})(\overrightarrow{v_{T_i^0}} + \overrightarrow{v_{T_i^1}} + \overrightarrow{v_{T_i^2}})^T) \tag{15}$$

M_F is a 3×3 symmetric matrix. According to Linear Algebra, M_F can be diagonalized under an orthonormal basis. The basis consists of three eigenvectors. According to the eigenvalues of M_F, the feature region can be classified as belonging to one of the following three categories.

- Three nearly equal eigenvalues (spherical).
- One eigenvalue much larger than the other two (long ellipsoid).
- Two nearly equal eigenvalues that are much larger than the third (flat ellipsoid).

Let α, β, γ be the three eigenvalues of M_F after normalization such that $\alpha^2+\beta^2+\gamma^2 = 1$. Let $C = \{(\alpha, \beta, \gamma) | \alpha^2 + \beta^2 + \gamma^2 = 1, \alpha, \beta, \gamma \geq 0\}$ be the set of all valid configurations. It is a spherical triangle in the first octant. There are seven special configurations that correspond to the three long ellipsoids, the three flat ellipsoids, and the perfect sphere. By building the

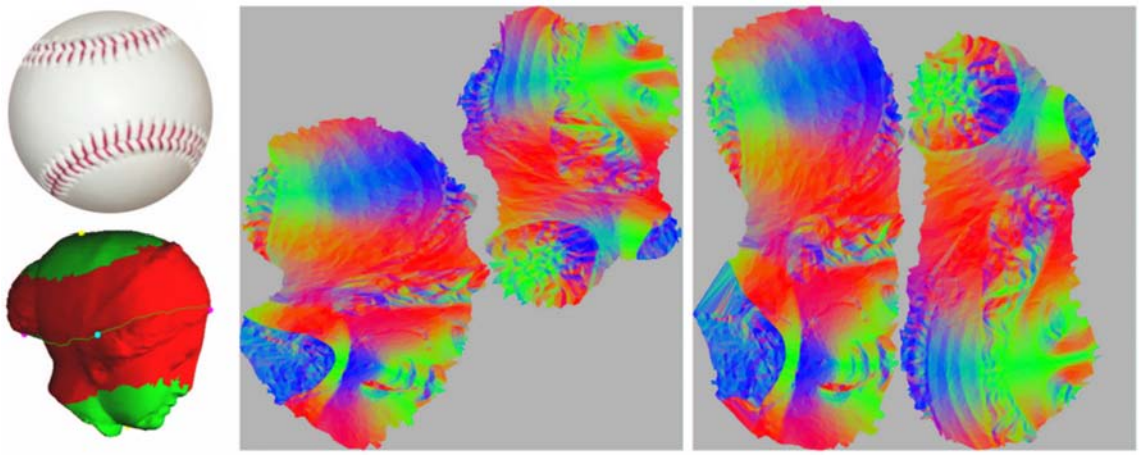

Figure 13: The "baseball" decomposition of the Venus model (lower-left) and the corresponding normal maps from patch unfolding with different stretch metrics: Sander's metric [78] (left) and the Green-Lagrange tensor (Section 4.2, right).

Voronoi regions on C using spherical distance, I classify every shape based on the position of its configuration in C. Alternatively, one can use the classification measure proposed by Kindlmann and Weinstein [48], which produces similar classification for the test models that I use.

In the case of a long ellipsoid, I find a pair points $(\mathbf{p}, \mathbf{q})$ on F such that they achieve maximum surface distance. This can be approximated by letting $\mathbf{p}$ be a global maximum of AGD and letting $\mathbf{q}$ be the point that is furthest away from $\mathbf{p}$ on the surface. I then find the shortest path γ between $\mathbf{p}$ and $\mathbf{q}$ and cut the surface along γ by duplicating all of its vertices except $\mathbf{p}$ and $\mathbf{q}$. This converts the surface into a single patch (topological disk).

For the flat ellipsoid case, I first identify an eigenvector associated with the smallest covariance eigenvalue. Then I find the two most distant surface points x_1 and x_2 along this vector in opposite directions away from the surface's center μ. Using region growing, I find the Voronoi regions for x_1 and x_2. Both regions are homeomorphic to a disk.

In the case of a sphere, we could treat it as a flat ellipsoid and create two patches that are much like hemispheres. Unfolding such patches causes high stretch. Instead, I use an approach that is inspired by the two identical patches of a baseball (see Figure 13, upper-left). I construct these regions based on two C-shaped curves, each of which travels half the way around each of the two mutually perpendicular great circles. To compute these curves,

I find the three pairs of antipodal points on the surface that passes through the surface center μ, along the three eigenvector directions. Call these points $x_1, x_2, y_1, y_2, z_1, z_2$. One C-curve passes through x_1, y_1, x_2, and the other connects z_1, y_2, z_2. Using region growing, I compute the "baseball decomposition" of the surface by building the surface Voronoi regions corresponding to the C-curves. The lower-left portion of Figure 13 shows one of these curves and the corresponding Voronoi regions (red and green) for the Venus model. Also shown in the same figure are normal maps of corresponding to the decomposition obtained by using two different patch unfolding methods: Sander's metric [78] (middle), and the Green-Lagrange tensor (Section 4.2, right). As will be discussed in Section 4.2, patch unfolding using the Green-Lagrange tensor results in less overall stretch.

Sometimes a feature is a curved long cylinder, such as the feline's tail, whose covariance analysis is similar to that a flat ellipsoid or a spherical surface. In this case, the center μ is situated outside the volume enclosed by the surface and not all three pairs of antipodal points can be found. When this happens, I simply treat the surface as a long ellipsoid.

CHAPTER IV

PATCH UNFOLDING AND PACKING

The second stage in the parameterization pipeline is *patch unfolding*, which is responsible for laying out every surface patch onto a plane without any foldovers. In addition, stretch should be minimized so that an even sampling rate is available across the surface.

The unfolding problem has its root in map-making, in which a portion of the Earth surface is "projected" onto a piece of paper. It is crucial to design the projection (unfolding) such that surface distance, area and angle (direction) are faithfully maintained in the map. I will review relevant background from classical differential geometry in the next section, and describe my unfolding technique. An excellent tutorial and survey for surface unfolding techniques can be found in [25].

4.1 Background

A surface parameterization for a 3D patch S is a diffeomorphism P between S and a region in the plane $P^{-1}(S) \subset \{(s,t) \mid s,t \in \mathbb{R}^2\}$, i.e., P is bijective, and both P and P^{-1} are differentiable. P is an *isometry* if it maintains distances between S and $P^{-1}(S)$, that is, $d(P(x), P(y)) = d(x,y)$ for all points $x, y \in S$. An ideal surface parameterization is an isometry, which means an even sampling rate is possible based on P.

Measuring angles and distances on a curved surface depends on the *first fundamental form* of the surface. Suppose $S = \begin{pmatrix} X(s,t) & Y(s,t) & Z(s,t) \end{pmatrix}^T$. The partial derivatives of S are:

$$S_s = \begin{pmatrix} \partial X/\partial s & \partial Y/\partial s & \partial Z/\partial s \end{pmatrix}^T, \quad S_t = \begin{pmatrix} \partial X/\partial t & \partial Y/\partial t & \partial Z/\partial t \end{pmatrix}^T \tag{16}$$

Any smooth curve $\gamma \subset S$ can be parameterized as

$$\gamma(r) = \begin{pmatrix} X(s(r),t(r)) & Y(s(r),t(r)) & Z(s(r),t(r)) \end{pmatrix}^T \tag{17}$$

for some $r \in \mathbb{R}$. The total differential of γ is $d\gamma = S_s ds + S_t dt$. The inner product between two curves γ_1 and γ_2 at their intersection is:

$$d\gamma_1 \cdot d\gamma_2 = \begin{pmatrix} ds_1 & dt_1 \end{pmatrix} \begin{pmatrix} E & F \\ F & G \end{pmatrix} \begin{pmatrix} ds_2 \\ dt_2 \end{pmatrix} \tag{18}$$

where $E = S_s \cdot S_s, \quad F = S_s \cdot S_t, \quad G = S_t \cdot S_t$. Equation 18 is the first fundamental form for the surface S. Furthermore, $g = \begin{pmatrix} E & F \\ F & G \end{pmatrix}$ is the *metric tensor*, and it allows us to measure distances, angles, and areas on surfaces. We have the following results from classical differential geometry. The parameterization P is an isometry if and only if the metric tensor is the 2×2 identity matrix everywhere. P is conformal ("angle-preserving") if the metric tensor is a scaling, and P is equiareal ("area-preserving") if the metric tensor has a determinant of ± 1. Clearly, an isometry is also conformal and equiareal. In addition, a conformal and equiareal mapping is an isometry. Except for developable surfaces, most patches do not have an isometric parameterization. This is due to the fact that these surfaces have different first fundamental forms than that of a plane. On the other hand, it is always possible to construct conformal or equiareal maps for a patch.

In computer graphics, surface parameterizations are often constructed for triangular meshes. In this case, metric tensors are constant within each triangle. Classical unfolding techniques seek to produce either conformal or equiareal maps. A popular approach approximate conformal mappings by computing *discrete harmonic maps* [18, 22, 53]. In this approach, the boundary vertices of the patch are first assigned to the boundary of a convex polygon in the parameterization domain. Next, the texture coordinates of the interior vertices are determined by solving a spare linear system. These methods are fast and stable, and the solution is unique [53, 15]. However, conformal mappings do not preserve areas. Unfortunately, regions can be stretched or compressed, causing uneven sampling rates. Sander et al. [78] propose a post-processing step in which the texture coordinates of the interior vertices of a patch are optimized to reduce a form of geometric stretch (which I will refer to as *Sander's stretch metric*), and their work inspired my own stretch optimization. I seek a definition of stretch that balances between conformal and equiareal mappings. In

addition, I allow the optimization of the boundary vertices of the patch by adding scaffold triangles. This technique can be applied to any parameterization to further reduce stretch.

4.2 The Green-Lagrange Tensor: a Balanced Stretch Metric

Sander's stretch metric (used in [78, 53]) helps balance the sampling given by the parameterization. Unfortunately, it does not distinguish between isometry and anisotropic stretch. To illustrate this point and to introduce my new balanced stretch metric, I review Sander's metric and related background.

For a triangle $T = \{p_1, p_2, p_3\}$ in the surface $S \in \mathbb{R}^3$ and its corresponding texture coordinates $U = \{u_1, u_2, u_3\}$ in $\mathbb{R}^2 =< s, t >$, the parameterization $P : U \rightarrow T$ is the unique affine mapping that maps u_i to p_i. To be more specific, let $A(a, b, c)$ be the area of the triangle formed by vertices a, b and c, then

$$P(u) = \frac{A(u, u_2, u_3)p_1 + A(u_1, u, u_3)p_2 + A(u_1, u_2, u)p_3}{A(u_1, u_2, u_3)} \tag{19}$$

The metric tensor induced by this mapping is:

$$G = \begin{pmatrix} P_s \cdot P_s & P_s \cdot P_t \\ P_s \cdot P_t & P_t \cdot P_t \end{pmatrix} = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \tag{20}$$

The eigenvalues of G are:

$$\{\gamma_{max}, \gamma_{min}\} = \sqrt{\frac{(a + c) \pm \sqrt{(a - c)^2 + 4b^2}}{2}} \tag{21}$$

which represents the maximal and minimal stretch of a non-zero vector. Sander's metric is defined as the average stretch metric in all possible directions, i.e.,

$$L^2(T) = \sqrt{(\gamma_{max}^2 + \gamma_{min}^2)/2} = \sqrt{(a + c)/2} \tag{22}$$

The metric has a lower bound 1 and isometries achieve this lower bound.

Equation 19 assumes that the area of the triangle equals the area of its textural image. When computing the global stretch, it is assumed that the total area of the surface patch equals the total area of the textural image. This means that we need to add this global scale factor to each triangle.

$$\rho = \sqrt{\frac{\sum_{t \in S} A(t)}{\sum_{t \in S} A'(t)}} \tag{23}$$

where $A(t)$ is the surface area of triangle t and $A'(t)$ is the area of the textural image of t. Then Equation 19 can be rewritten as:

$$P(u) = \frac{A(u, u_2, u_3)p_1 + A(u_1, u, u_3)p_2 + A(u_1, u_2, u)p_3}{\rho A(u_1, u_2, u_3)} \tag{24}$$

Notice that individual triangles in general have different scale factors from the global one. Unfortunately under this scenario, there are other mappings between a triangle and its textural image that are not isometries for which the measure also gives the value of one. In particular, this metric cannot distinguish anisotropic stretch. For instance, all three of these tensors:

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0.5 & 0 \\ 0 & 1.5 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \tag{25}$$

result in the same stretch measured in Sander's metric, but the first one is clearly the most desirable. For this reason, I use the *Green-Lagrange tensor* to measure stretch and to guide patch optimization. Using the Green-Lagrange tensor as the stretch metric has been proposed before [57]. However, it has not been used for patch optimization.

Using the above terminology, the *Green-Lagrange tensor* of G_t is defined as $\| G_t - I \|$, in which I is the identity matrix. The square of the *Frobenius norm* of this tensor is

$$T(G_t) = (\| G_t - I \|_{\mathbf{F}})^2 = (a-1)^2 + 2b^2 + (c-1)^2 \tag{26}$$

It is zero if and only if G_t is an isometry. I therefore define the stretch as:

$$E_t^2 = 2T(G) = 2((a-1)^2+2b^2+(c-1)^2) = [(a-c)^2+4b^2]+[(a+c-2)^2] = E_{conformal}^2+E_{area}^2 \tag{27}$$

Notice that for the tensor G to be conformal, we need $a = c$ and $b = 0$. When these conditions are met, G becomes a scaling of magnitude $a = c$. G is an isometry if and only if $a = c = 1$. The metric E_t seeks a balance between angle preservation $E_{conformal}^2$ and

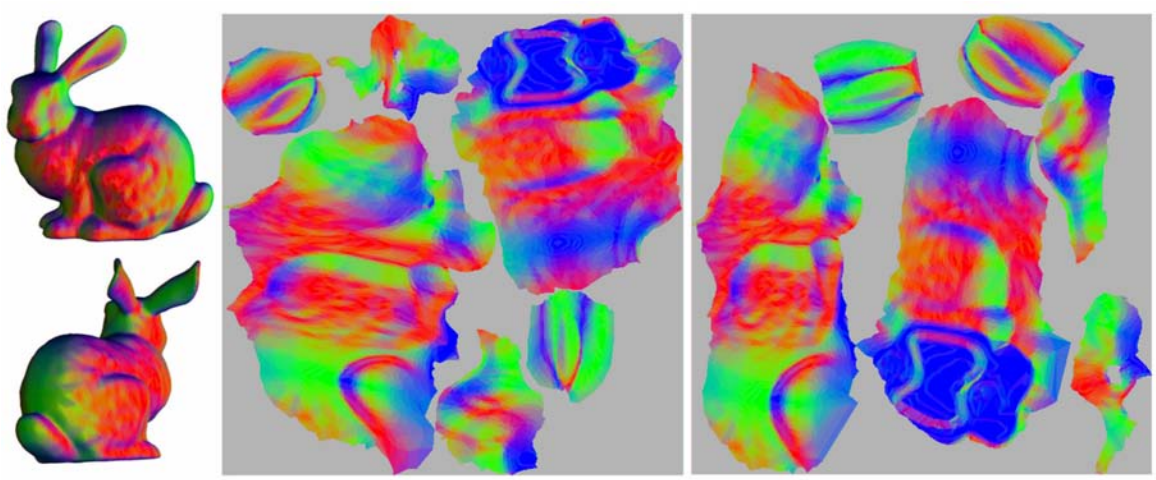

Figure 14: This figure shows the surface parameterization of the bunny obtained by vertex optimization based on Sander's stretch metric [78] (middle) and the Green-Lagrange tensor (right). Optimization based on Sander's metric causes high anisotropic stretch, especially the two largest patches. On the other hand, optimization based on the Green-Lagrange tensor performs well for all patches. Compare the tail and the two rear legs (the square bumps on each side).

area preservation E_{area}^2. A triangle's mapping is an isometry if and only if $E_t = 0$. This metric distinguishes between anisotropic stretch and isometry. In addition, it penalizes both undersampling and oversampling. However, the penalty is more severe for undersampling. This is desirable for texture mapping when a global isometry is not available. Note that Sorkine et al. [85] devised a different stretch metric that also distinguishes anisotropic stretch from isometry. I choose not to use their metric because it uses a max function, causing it to give equal stretch values to some cases that I feel should be distinguished.

The total balanced stretch of a patch S is therefore,

$$E^2(S) = \sum_{t \in S} E_t^2 = \sum_{t \in S} \{[(a_t - c_t)^2 + 4b_t^2] + [(a_t + c_t - 2)^2]\} \tag{28}$$

The ideal value $E(S)$ for a patch S is zero, meaning all triangles in the patch are mapped isometrically.

Figure 14 compares the unfolding of the bunny surface using Sander's metric (middle) and the Green-Lagrange tensor (right). Notice that on the two largest patches, unfolding with Sander's metric produces anisotropic stretch (the tail and the two rear legs). On the other hand, the Green-Lagrange tensor performs well on all patches. Figure 13 shows the

same comparison for the Venus model. Again, optimization using Sander's metric causes anisotropic stretch. In Section 5.2, I will show the Green-Lagrange tensor also performs better in terms of image fidelity, despite sometimes having lower packing efficiencies.

4.3 Boundary Optimization with Scaffold Triangles

The process of *patch optimization* refers to moving vertices in the plane to minimize a given stretch metric. Most patch optimization methods handle boundary vertices of a patch differently from interior vertices. For initial layout, the boundary vertices are typically either mapped to the vertices of a convex polygon, or solved through conformal mapping. Sander et al. [78] perform optimization by going one by one through the interior vertices and making local changes. They check whether moving a given vertex along a randomly chosen line will improve the stretch of the incident triangles of the vertex. I adopt a similar optimization strategy, with one modification that I describe below. During optimization there could be *global foldovers*, in which the textural images of two triangles overlap even though they are relatively far away from each other on the surface. Collision detection in the texture domain is therefore needed to prevent this from happening [77].

I introduce a new optimization method that allows the boundary vertices to move freely without the need for checking for global foldovers. First, an initial harmonic parameterization is obtained by mapping the patch boundary to a planar convex polygon and by solving a linear system for the interior vertices (Figure 15, (b)). This step is essentially the same as [22]. Next, I construct a "virtual boundary" (a square) in the parameterization plane that encloses the patch. The 3D coordinates of the square are assigned to be mutually different and outside the convex hull of the patch in the 3D space. As we will see next, the exact coordinates of the virtual boundary are insignificant provided they do not coincide with each other or with the patch. Scaffold triangles are used to triangulate the region between the original patch boundary and the virtual boundary (See Figure 15, (c)). Finally, I perform patch optimization [78] on the "enlarged" patch using the Green-Lagrange tensor (Section 4.2). There are two issues about scaffold triangles that need attention.

1. How to define stretch for scaffold triangles?

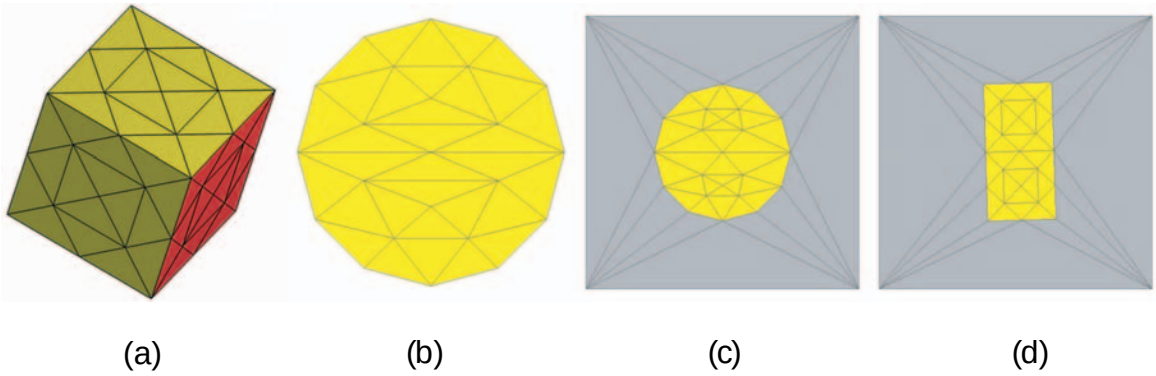

Figure 15: This figure demonstrates the effect of scaffold triangles on patch unfolding. In (a), a patch on the cube is colored in yellow. In (b), this patch is unfolded without using scaffold triangles. The image in (c) shows the unfolding of the same patch with scaffold triangles but without optimization, and the image in (d) corresponds to unfolding with scaffold triangles and optimization. In (c) and (d), scaffold triangles are colored in gray. When both optimization and scaffold triangles are used, the patch is unfolded with zero stretch.

2. How to define and maintain their connectivity?

The first issue is handled as follows: the stretch of a scaffold triangle is defined as infinity if there is a foldover, otherwise it is defined as zero. This allows a patch boundary vertex v to move within its immediate incident triangles to obtain better stretch without the need to checking for global foldovers. Furthermore, the exact 3D coordinates of the virtual boundary are insignificant.

The second issue appears when the initial connectivity of scaffold triangles unnecessarily constrains the movements of boundary vertices. This is because scaffold triangles are designed to prevent global foldovers, i.e., one patch vertex "walks" onto a patch triangle other than its immediate neighboring triangles, which unfortunately include the scaffold triangles. To remedy this overly conservative approach, I allow the scaffold regions to be retriangulated at the end of each optimization iteration, i.e., when all the vertices (including the boundary vertices) have been moved. The retriangulation is achieved by performing edge flips on all the edges that are adjacent to two scaffold triangles if the operations improve the triangles' aspect ratios. Figure 16 demonstrates the effectiveness of the retriangulations when optimizing a patch on the feline. The image in (a) shows the initial scaffold triangle connectivity. Optimization without changing the connectivity reduces stretch to 2.12 and

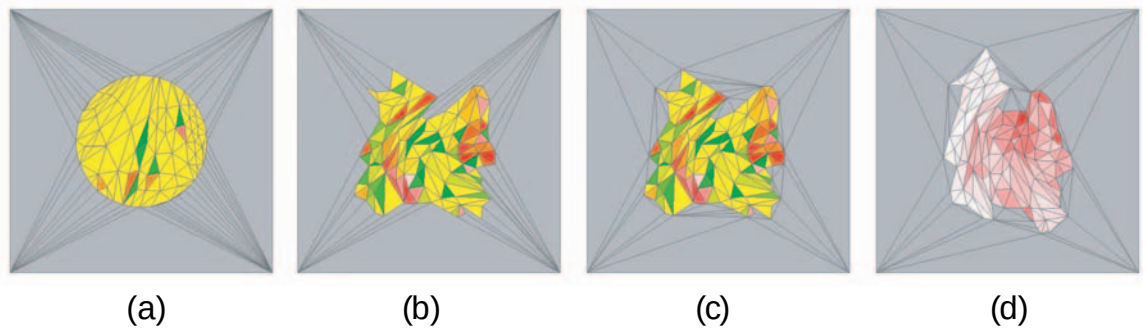

Figure 16: Remeshing scaffold triangles can help further reduce stretch. The image in (a) illustrates the initial parameterization for a patch on the feline model. With the initial scaffold triangle connectivity, the optimization reduces stretch to 2.12 (b) and stops. After retriangulation (c), optimization further reduces stretch to 0.10 (d). Colors indicate stretch: white $0-0.16$, red $0.16-0.33$, yellow $0.33-0.5$, green $0.5-0.66$. Stretch is measured using the Green-Lagrange tensor.

stops (b). After retriangulation (c), optimization further reduces stretch to 0.10 (d). Colors indicate the amounts of stretch, with white/red indicate low stretch, and yellow/green indicate high stretch. Stretch is measured using the Green-Lagrange tensor.

The image in Figure 15 (d) shows the result of optimization with scaffold triangles, which is in contrast to (c) (optimization without scaffold triangles).

The shape of the virtual boundary and the connectivity of the scaffold triangles are insignificant since they merely serve as a placeholder to allow the boundary vertices of the patch to move freely without causing global foldovers. This is very different from the work by Lee et al. [52], in which virtual boundaries are constructed as part of the springs for obtaining initial parameterization. In their work, the shape and the connectivity of the virtual boundaries directly affect the stretch of the resulting parameterization. Indeed, several layers of virtual boundaries are often required to produce reasonable results using their method. In my work, only one layer is needed.

Scaffold triangles also come from hole-filling operations that occurred during genus reduction and feature identification. The vertices of the hole-filling scaffold triangles are allowed to move within their 1-ring neighborhoods just like other interior vertices. These scaffold triangles do not contribute to the stretch metric, and edge flips are allowed. Several of the patches in the texture maps shown in Figure 17 (right column, dinosaur) and

Figure 20 (bottom row, dragon) have such holes that make use of scaffold triangles.

4.4 Patch Packing

The final step of surface parameterization is *patch packing*, which refers to placing the unfolded patches inside a rectangular region (texture map) without causing any overlaps between the patches. The ratio between the total space occupied by the patches and the area of the rectangle is the *packing efficiency*. Higher packing efficiency indicates less wasted space in the texture map. The problem of finding optimal packing is a special instance of a NP-hard problem: *containment and minimum enclosure*. The problem has been studied extensively in the textile industry and the computational geometry community [59].

Several packing algorithms have been proposed as parts of some surface parameterization techniques. Sander et al. [78] avoid overlap between patches by approximating them with their convex hulls. Starting with an empty rectangular region, the patches are placed one at a time starting from the lower-left corner of the region. A patch is placed next to its immediate predecessor in a horizontal sweeping fashion. They keep track of a "horizon", i.e., a height function for every column in the region. If there is not enough space, the next patch is placed above the horizon. This method is fast for a large number of small patches, but it often results in low packing efficiency since the patches are estimated by convex polygons.

Lévy et al. [53] propose an improvement in which patches are no longer approximated by their convex hulls. Furthermore, the horizon is better maintained so that more spaces are available to pack small patches. This method results in better packing efficiency. However, since it still makes use of a horizon, empty spaces beneath the horizon are not utilized.

These methods work well if the number of patches is large. Since my patch creation technique tends to produce a small number of large and often elongated patches, I develop a packing technique that takes into account the orientations of the patches and in general achieves better packing efficiency than existing packing techniques [78, 53]. Later I discover that this method is very similar to the packing technique by Sander et al. [79]. Unlike traditional packing method and my method, they do not assume a square texture domain.

My packing algorithm is based on the following two observations. First, the feature-based patch creation method I have described tends to produce a small number of patches. Second, several of these patches are large and have elongated shapes. The first observation enables us to perform optimal searching that would have been impractical for patch creation methods that produce hundreds of patches. The second observation indicates that the orientations of the large and elongated patches can help create gaps, inside which smaller patches can be placed.

My algorithm starts by creating a *canvas*, i.e., a rectangular grid cell structure with the textural resolution. Each cell is initially assigned *unoccupied*. With the same resolution, I discretize the bounding box of each patch's textural image into rectangular grid cells. A cell is *occupied* if it intersects the textural image with at least one triangle in the patch. Otherwise, it is *unoccupied*. For every patch, I obtain eight variations of grid cells by the combination of reflections with respect to its vertical axis, the horizontal axis, and the diagonal.

I iteratively insert one patch into the canvas at a time in the decreasing order of the patch's textual size (area). Let N be the desired size of the final texture map. Initially, all $N \times N$ grid cells in the canvas are *unoccupied*. The first patch is placed at the lower-left corner of the canvas. After one patch is inserted, some grid cells in the canvas will be *occupied*. When inserting the next patch P_i, its eight variations are examined to find the one that minimizes *wasted space* in the canvas. To be precise, let α, a $m \times n$ grid cells, be a variation for P_i. I wish to place the lower-left corner of α in the (a, b) grid cell in the canvas such that the following conditions are met.

1. For any *occupied* grid cell (p, q) in α, the corresponding grid cell $(a + p, b + q)$ in the canvas is *unoccupied*.
2. α minimizes $\max(a + m, b + n)$.

In other words, we wish to place the patch as close to the lower-left portion of the canvas as possible. Once the best variation is chosen, I translate and scale the patch textural image to reflect its position and orientation in the canvas.

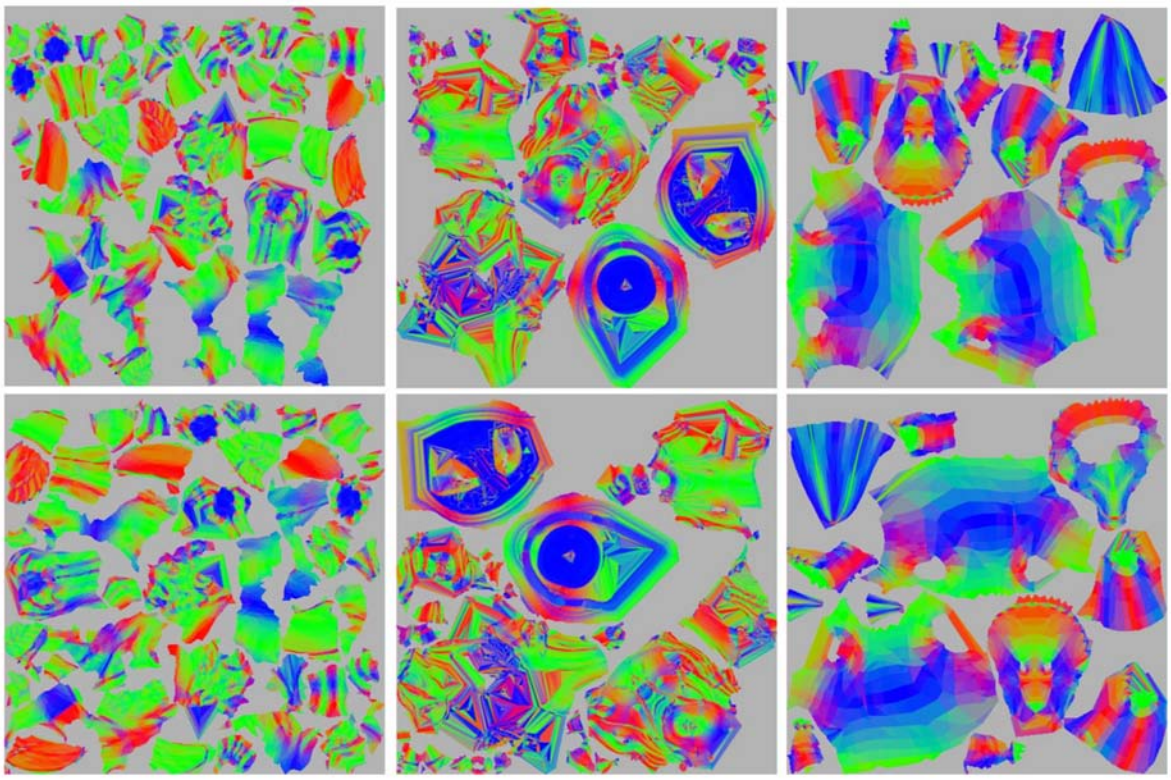

Figure 17: This figure compares the packing results based on the algorithm of Lévy et al. [53] (top row) and my algorithm (bottom row) for three test models: the feline (left), the Buddha (middle), and the dinosaur (right). Notice that with my algorithm, the spaces under the "horizon" are used to pack smaller patches, and some patches are rotated/reflected to achieve tighter packing.

When all patches have been inserted, usually only $M \times M$ grid cells in the canvas become occupied. For all the test models, M is between one-third and one-half of N, the size of the canvas. Therefore, I perform scaling to all patches with the same factor so that $M \times M$ grid cells are mapped to $[0, 1] \times [0, 1]$.

Figure 17 shows the improvement of my packing algorithm (bottom row) over the algorithm by Lévy et al. [53] (top row) for three test models: the feline (left), the Buddha (middle), and the dinosaur (right). Notice that with my algorithm the space under the "horizon" is reused to pack small patches, and patches are rotated/reflected to achieve tighter packing.

CHAPTER V

RESULTS AND PARAMETERIZATION ERROR METRIC

5.1 Results

I have applied my feature-based surface parameterization method to a number of test models. The results for the bunny and the dinosaur are shown in Figure 2 and 6, respectively. Figure 18 shows the results of some other 3D models, including three surfaces with non-zero genus (the Buddha, the dragon, and the feline). Notice that in general the created features are intuitive. For example, the horns and legs of animals are segmented from the bodies, and the Buddha's stand is identified as a single feature (a flat ellipsoid). Figure 19 compares the feature segmentation results for models at different resolutions (lower-resolution models are shown in the top row, and higher-resolution models in the bottom row). Notice that the segmentation results are similar between models of different resolutions, except for the Buddha. The torso of the Buddha is divided into two regions for the lower resolution model, while for the higher-resolution model the torso is a single feature. (I wish to emphasize that no real animals were harmed during this research.)

Figure 13 (right) and Figure 17 (lower middle) show the normal maps of the Venus and the Buddha respectively. In a normal map, colors are used to encode unit surface normal [78]. Because of the many sharp creases in the Venus and the Buddha, patch creation methods based on surface curvature would have split the surfaces into many tiny patches. However, the feature-based segmentation method that I describe in this thesis was able to create large patches with little stretch.

Figure 20 shows textured models (left column) and the corresponding texture maps (right column) of the Buddha (top), the feline (middle), and the dragon (bottom). Table 1 provides the average stretch for the patches of the test models, the feature creation times,

model name	# polygons	# patches	stretch (Green-Lagrange)	patch creation time	patch unfolding time
Buddha	20,000	28	1.56	6:32	27:29
Buddha (large)	100,000	16	1.27	39:25	168:43
bunny	10,000	6	0.23	1:07	8:28
cow	10,524	29	0.28	2:15	4:01
dinosaur	10,636	14	0.25	1:37	5:53
dragon	20,000	24	0.83	3:00	16:23
dragon (large)	100,000	39	0.49	38:00	124:32
feline	10,000	41	0.22	2:31	2:32
feline (large)	100,000	46	0.29	32:57	109:27
horse	10,000	27	0.22	1:30	3:21
Venus	10,000	2	0.17	0:11	11:38
rabbit	10,000	8	0.24	0:53	4:50

Table 1: Average stretch (measured in the Green-Lagrange tensor) and timing results (minutes:seconds) for feature segmentation and patch unfolding using the techniques described in this thesis. Times are for a 2.4 GHz PC.

and the patch unfolding times using my method. The texture used for the Buddha is a wood texture from Perlin's noise [66]. The textures used for the feline and the dragon were created by performing example-based texture synthesis directly on the surfaces [93, 97].

5.2 An Image-Based Quality Measure for Surface Parameterization

Measuring the quality of a particular surface parameterization is an important yet complicated issue. It has several components.

1. Stretch: affects the sampling rate across the surface.
2. Seams: cause discontinuities across the patch boundaries.
3. Smoothness: measures the amount of sharp changes in the sampling rate across interior patch edges.
4. Packing efficiency: measures the percentage of pixels in the texture map that correspond to some part of the surface ("useful pixels").

When evaluating a surface parameterization method, it is *not* clear how these components should be combined to give an indication to the quality of the resulting map. On

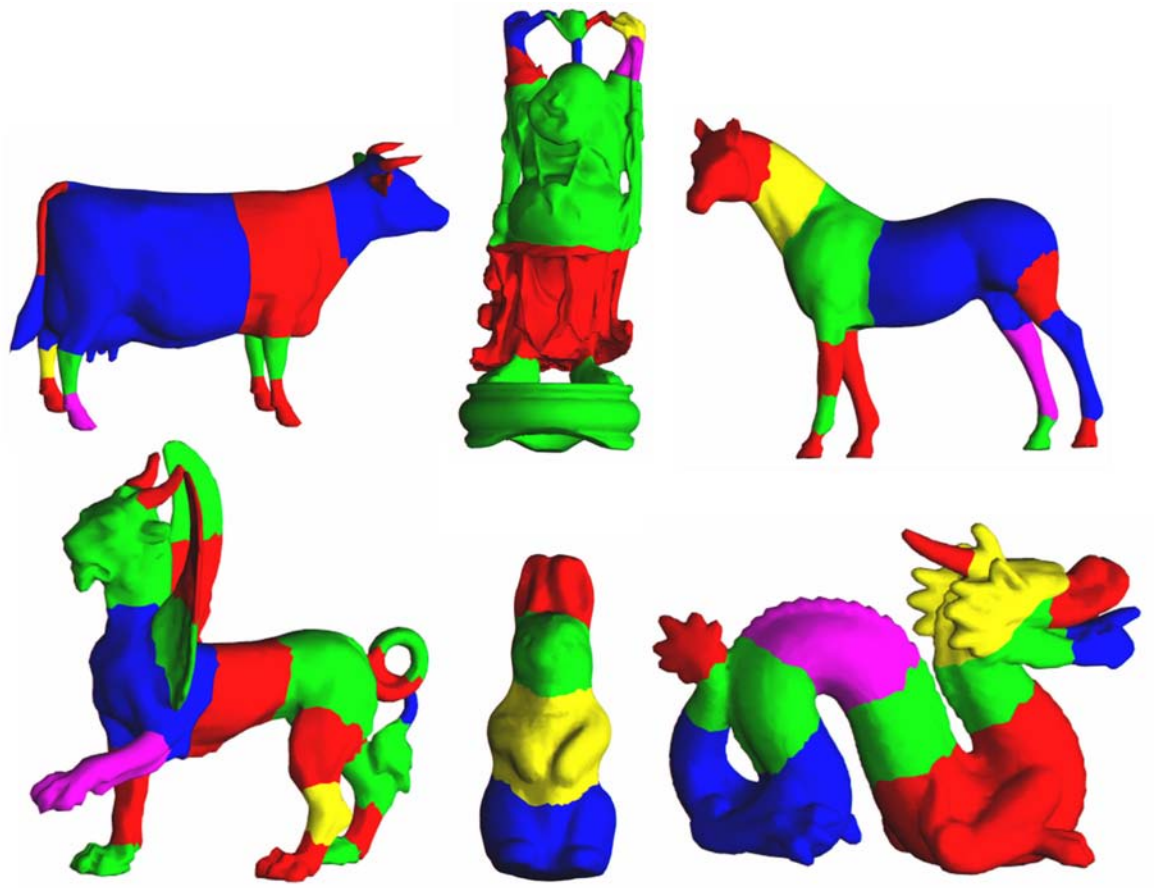

Figure 18: This figure shows the result of my feature segmentation method on various test models. The cow, the horse and the rabbit model are genus zero surfaces. The genus of the original models for the dragon, the happy Buddha, and the feline are one, six, and two, respectively.

the other hand, for texture mapping applications, the quality of a surface parameterization should reflect "image fidelity", i.e., the faithfulness of the images produced using texture maps to the images in which surface signals are directly computed. I will describe an image-based metric, which draws inspiration from the work on image-driven mesh simplification [55].

To be more specific, given a surface parameterization P, I first compute a continuous and smooth surface signal and store the result in a texture map based on P. Then, I render the surface from many viewpoints using the texture map and compare the image differences with respect to the true surface signal. In practice, I choose 20 orthographic viewpoints that are the vertices of a surrounding dodecahedron. Let M_0 be the surface with the signal

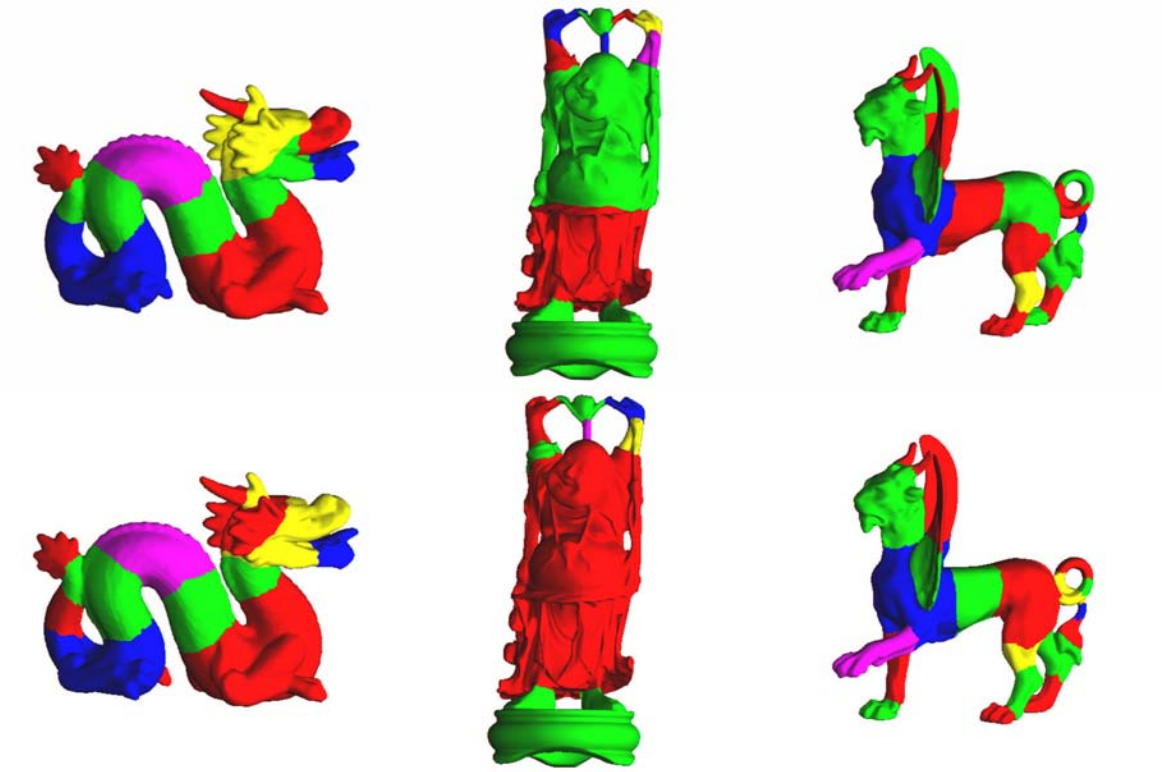

Figure 19: Segmentation results for three models at different resolutions (Top row: 10,000 for the feline and 20,000 for the Buddha and the dragon. Bottom row: 100,000 for all three models). Notice the segmentation results are very similar for the models except for the Buddha, where the torso is divided into two regions for the lower-resolution model, and remains one patch for the upper-resolution model.

directly computed, and M_i be the textured surface with the size of the texture map being $2^i \times 2^i$. The RMS "image" error between the images is calculated as:

$$RMS\left(M_i, M_0\right) = \sqrt{\sum_{n=1}^{20} D_i^n} \tag{29}$$

where D_i^n is the squared sum of pixel-wise intensity difference between the n-th image of M_i and M_0. Equation 29 can be seen as a discretized version of the following functional:

$$E(M_i, M_0) = \sqrt{\frac{\int_{\mathbf{p}\in S} D^2(M_0(\mathbf{p}), M_i(\mathbf{p}))V(\mathbf{p})d\mathbf{p}}{\int_{\mathbf{p}\in S} V(\mathbf{p})d\mathbf{p}}} \tag{30}$$

Here $D^2(M_0(\mathbf{p}), M(\mathbf{p}))$ is a perceptual metric between colors. For this application, I use the following metric:

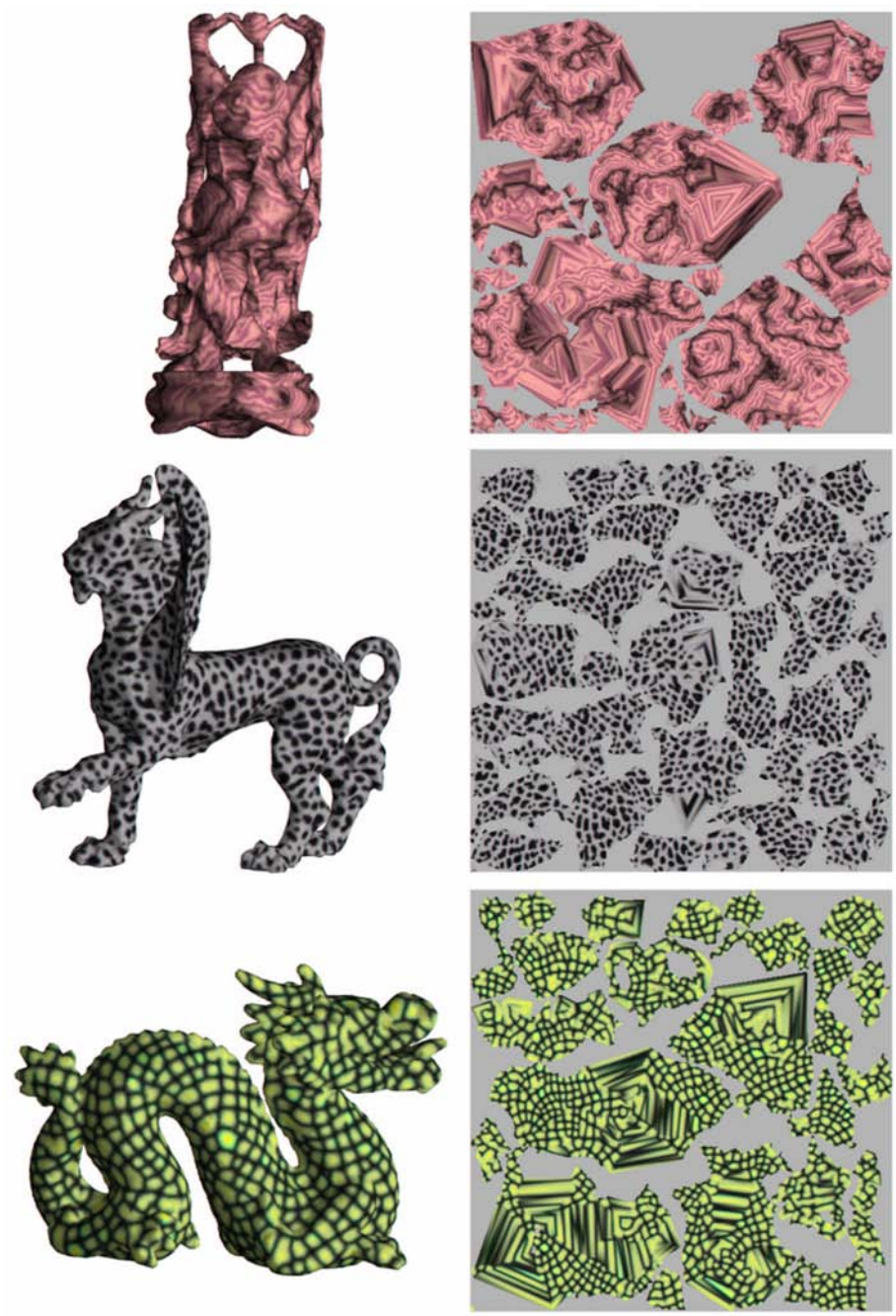

Figure 20: This figure shows the parameterization of three models using my feature-based algorithm. From top to bottom are: the Buddha, the feline, and the dragon. The images in the left column are textured models, while the images in the right are the corresponding texture maps (512×512). The genus of the original models for the Buddha, the feline, and the dragon are six, two, and one, respectively.

$$D((r_1, g_1, b_1), (r_2, g_2, b_2)) = \sqrt{(\frac{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2)}{3}} \tag{31}$$

$V(\mathbf{p})$ is the view-independent surface visibility as defined in [102], and it measures the visibility of a point $\mathbf{p}$ in the surface with respect to the viewpoints on a surrounding sphere of infinite radius. Therefore, $E(M_i, M_0)$ takes into account of surface visibility in addition to color errors caused by stretch, seams, packing efficiencies, and smoothness of the parameterization. As demonstrated in [55, 102], the error metric can be approximated by sampling a small number of viewpoints that are evenly spaced in the view space.

One possible ideal surface signal can be obtained by first spreading a set of evenly spaced points on the surface and building a smooth function that uses these points as the bases. However, it can be time consuming to produce such a function. On the other hand, notice that the $3D$ checkerboard pattern has the nice property that it is easy to compute and the biggest differential in frequencies in all directions is bounded. Although not perfect, it is nonetheless a good starting point. To make the signal continuous, I replace each "box" section with a "hat". The frequency in each main axial direction is the same. In practice, the frequency is set to 1/16 of the maximum side of the bounding box of the surface.

Table 2 compares two unfolding methods, optimization with Sander's metric and the Green-Lagrange tensor, for nine test models. Notice that optimization with the Green-Lagrange tensor produces lower stretch for all the test models. Furthermore, despite sometimes having lower packing efficiencies, optimization with the Green-Lagrange tensor produces lower image errors for all the test cases. Figure 21 provides the visual comparison between the original signal (bottom-middle), the textured model using optimization with Sander's metric (bottom-left) and the Green-Lagrange tensor (bottom-right) for the Buddha model with the texture map of size 128×128. Notice the different level of blurring in the left image (front body and base) due to an uneven sampling rate. This phenomenon is less noticeable in the right image, which is based on optimization using the Green-Lagrange tensor. Compare their corresponding texture maps: left row (Sander's metric) and right row (the Green-Lagrange tensor). In Sander's technique, the patches created from the base are assigned more space than those from the Buddha's torso. This is not an issue using the Green-Lagrange tensor (right).

Comparison for patch unfolding with different optimization metrics (top row using Sander's metric [78], bottom row using the Green-Lagrange tensor)					
	Stretch measured in Sander's metric	Stretch measured in Green-Lagrange	Packing Ratio	Image Error 128×128	Image Error 256×256
Buddha	1.27 1.18	26.80 1.56	0.67 0.68	8.28%	10.77%
bunny	1.13 1.02	3.92 0.23	0.60 0.65	14.46%	10.93%
cow	1.11 1.03	3.07 0.28	0.73 0.65	1.89%	1.62%
dinosaur	1.07 1.03	1.55 0.25	0.59 0.66	13.22%	5.16%
dragon	1.26 1.13	13.78 0.83	0.67 0.67	11.14%	12.84%
feline	1.10 1.02	1.73 0.22	0.66 0.64	7.25%	3.46%
horse	1.09 1.03	1.65 0.22	0.67 0.66	5.57%	0.93%
Venus	1.10 1.02	2.99 0.17	0.59 0.66	16.29%	15.26%
rabbit	1.12 1.03	3.00 0.24	0.68 0.65	8.73%	4.54%

Table 2: This tables compares two stretch metrics for guiding Sander-style vertex optimization [78]. With the exception of the columns labeled "Image Error", the top row in each data cell are the results from using Sander's metric, and the bottom rows are the results from using the Green-Lagrange tensor (Section 4.2). For a comprehensive comparison, three measurements are provided: average stretch (the first two columns), packing efficiency (third column), and image-based error metric (the last two columns, Section 5.2). The numbers in the "Image Error" columns are the percentage of error difference of the image error caused by Sander's metric to the image error caused by the Green-Lagrange tensor. For all the test models, optimization with the Green-Lagrange tensor results in lower average stretch (either measured using Sander's metric or the Green-Lagrange tensor). Despite sometimes having lower packing efficiencies, optimization using the Green-Lagrange tensor results in less image errors for all the test models.

Now that I have presented a solution to the problem of surface parameterization, I will discuss the other half of this thesis: creating vector fields on mesh surfaces.

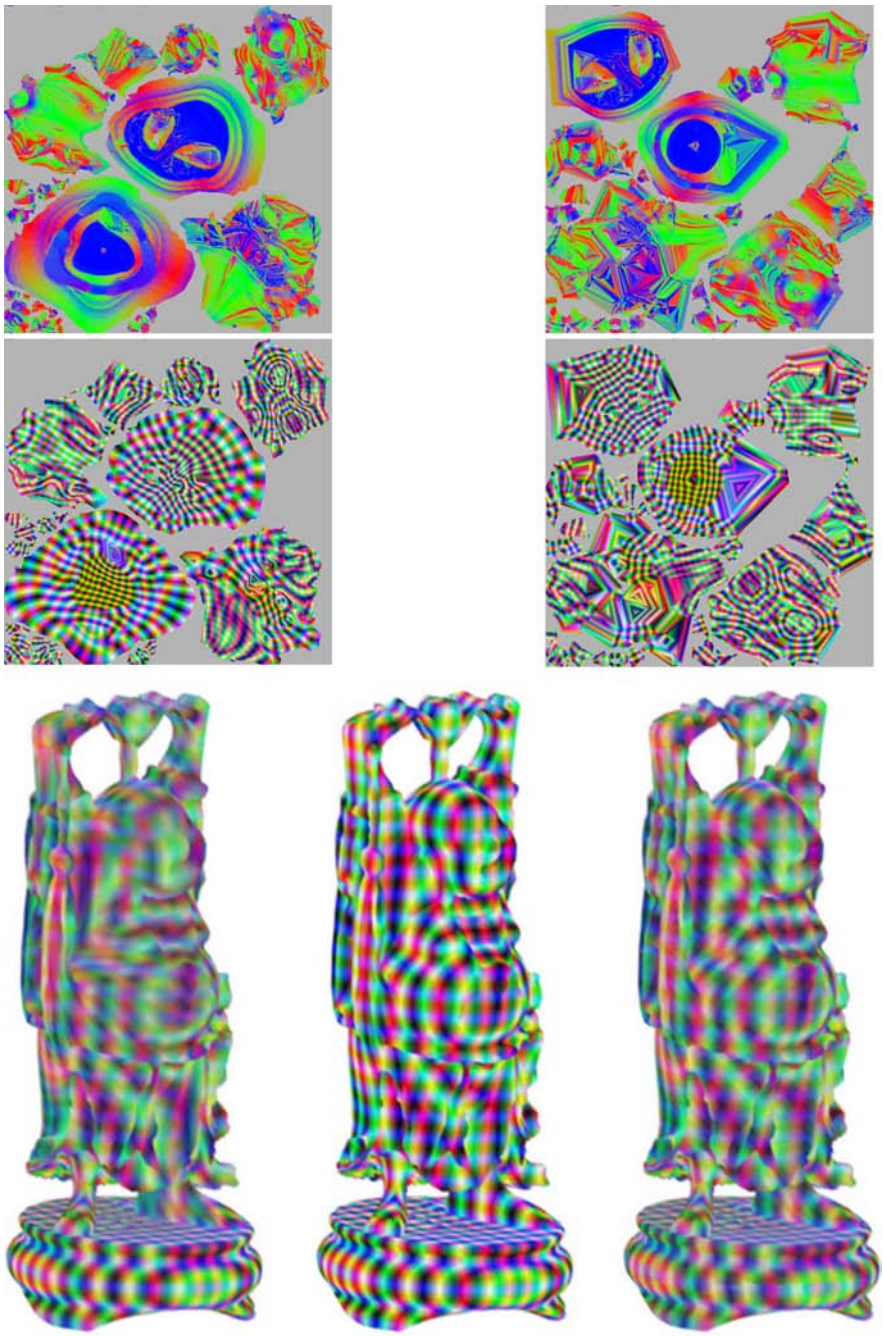

Figure 21: Comparisons between patch unfolding with Sander's metric [78] (left-column) and with the Green-Lagrange stretch tensor (right-column) for the Buddha model. Using the 3D texture described in Section 5.2 (original signal is shown in the middle of the bottom row). In each column, from top to bottom, are the normal map, the map of the 3D texture, and the textured model. Notice that in the left column (Sander's metric), the patches created from the base are assigned more space than those from the Buddha's torso. This uneven sampling rate results in a loss of signal in the textured model (Buddha's face, body, and feet, bottom-left). On the other hand, optimization using the Green-Lagrange tensor (right column) produces a more even sampling rate, and the reconstruction error is less noticeable.

CHAPTER VI

PREVIOUS WORK IN VECTOR FIELD DESIGN

Vector field *analysis* and *visualization* have been well studied, and a good survey is available in [35]. On the other hand, vector field *design* is far less explored. In this chapter, I will first survey the uses of vector fields and then review research related to vector field design.

Many graphics applications make use of an input vector field. For instance, in painterly rendering, the orientation of brush strokes are often based on the image gradient field [58, 38], which is a vector field. Similarly, in non-photorealistic illustration of surfaces, hatches are guided by a direction field, which can be inferred from a vector field. Girshick et al. [27] illustrate that principle curvature fields are best at conveying shapes. Hertzmann and Zorin [39] describe an efficient method for computing principle curvature fields for pen-and-ink illustration of smooth surfaces. In this thesis, I will show that vector field design allows a user to create vector fields that achieve different visual effects.

As we have seen in earlier chapters, example-based texture synthesis is an efficient way of creating textures over a 3D surface from an input image. However, for an anisotropic texture, the direction along which the input texture is copied onto the surface becomes an important issue. Praun et al. [70] divide the surface into a number of patches and define local frames for every patch based on an input vector field. The local frames allow the example textures to be copied with desired directions. They use texture blending near patch boundaries to minimize the artifacts caused by the discontinuity in the local frames. Turk [93] and Wei and Levoy [97] use a different technique that is based on neighborhood matching. Initially, a number of sample points are placed on the surface, and neighborhood relationships are established among them. Starting from a seed point, the system traverses over the sample points and computes their colors as follows. For a sample point, the colors of its neighbors are used to find the best match in the input image. The color of the pixel with the best match is then copied over to the sample point. In this approach, a vector

field is needed for two purposes: defining local frames, and determining the order in which the sample points are visited.

In fluid simulation, Stam [87] makes use of gravity or Coriolis force to simulate natural fluid flow over 3D surfaces. The external force in the system is a vector field, which can be designed to achieve different visual effects.

In vector field visualization, vector fields with known behaviors can be used to test existing visualization techniques. Van Wijk creates a vector field design system to test his *image-based flow visualization* technique for planar domains [94] and for curved surfaces [95].

Furthermore, a vector field design system can be used as a teaching aid. Imagine students learn important concepts about vector fields, such as vector field topology, through creating and manipulating a vector field and observing the changes.

As discussed in the Introduction (Chapter 1), a vector field design system should allow a user to create a wide variety of vector fields. In addition, for certain applications such as texture synthesis and non-photorealistic rendering, the singularities in the input vector field cause visual artifacts [70, 39]. Therefore, it is necessary for a vector field design system to provide control over vector field topology. In Figure 22, the singularities in the vector fields cause visual artifacts for texture synthesis (near the center of the bunny's tail and at the center of the feline's wing), and hatch-based illustration of surfaces (lower-left, the singularity underneath the chin). Through a topological editing operation, the user is able to fix the problem (compare with the lower-right image). Next, I will review existing vector field design systems for planar domains and for surfaces, respectively.

6.1 Vector Field Design Systems for Surfaces

There have been some prior work in creating vector fields on surfaces. In all the instances that I know, such systems have been created in a quick manner to generate vector fields for a particular application, such as texture synthesis [70, 93, 97], fluid simulation [87], or for testing a vector field visualization technique [95]. Furthermore, the details of these design systems have not been published.

There are three basic approaches for creating a surface vector field using these systems.

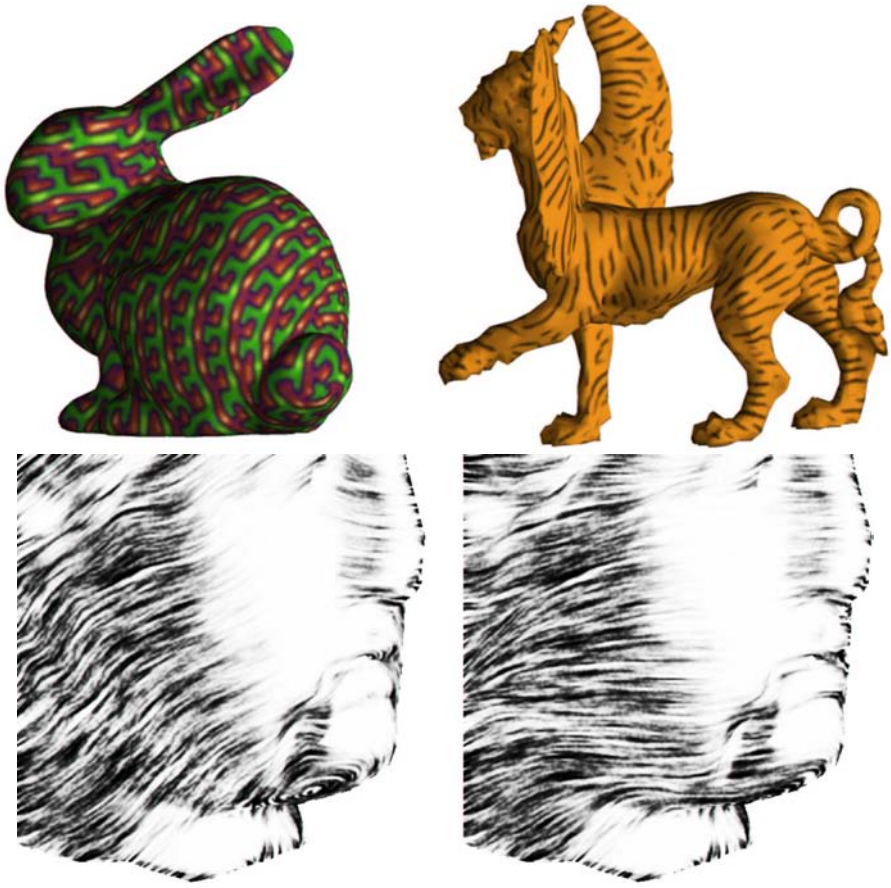

Figure 22: This figure highlights the need for the ability to control vector field topology. In the top row, the vector fields used for texture synthesis contain singularities, one at the center of the bunny's tail (upper-left), and another at the center of the feline's wing (upper-right). In both cases, the singularities cause the synthesis patterns to break up. In the bottom row, a singularity underneath the model's chin is clearly visible through the hatches (lower-left). In my system, the user has the ability to remove the singularities. Compare this image with the image in the lower-right, in which the singularity has been removed.

In the first approach, a 3D vector field is specified and projected onto the surface to obtain a tangential vector field. Stam [87] uses gravity or Coriolis force as the input external force to achieve certain physical phenomena. Van Wijk describes a vector field design system in which the user specifies arbitrary 3D basis vector fields, such as a constant vector field in a certain direction [95]. This way of creating surface vector fields is similar to performing texture synthesis on surfaces through solid textures. While it is simple and fast, achieving control is hard.

In the second approach, the user specifies desired vector values at a few locations on the

surface and the system performs relaxation to obtain a global surface vector field [93, 97]. This can be seen as a diffusion process in which the desired vector values are smoothly propagated to the rest of the surface. The diffusion process is achieved by solving a vector-valued Laplacian equation, and the resulting 3D vector field is then projected onto the surface.

In the third approach, Praun et al. [70] allow the user to specify vector values at a few places on the surface. Then a global vector field is constructed by interpolating these values using Gaussian radial basis functions over the surface.

These vector field design systems do not provide the user with control over vector field topology, such as the number and location of the singularities in the vector field. However, I borrow some of these ideas to create an initial vector field in the first of a three-stage design pipeline.

6.2 Vector Field Design Systems for Planar Domains

For planar domains, vector field design systems based on topological information have been demonstrated. Van Wijk creates a vector field design system to demonstrate his image-based flow visualization technique [94]. In his design system, the user specifies desired singularity locations and types. The system converts each specification into a basis vector field and combines them into a global vector field using radial basis functions. However, vector fields created in this manner often have more singularities than that the user intended. Because this system does not provide a way of removing undesired singularities, it lacks control over vector field topology.

Rockwood and Bunderwala [75] propose a technique in which the system uses geometric algebra to create vector fields based on user-specified singularity locations and types (source, saddle, etc). The user can interactively create a vector field by adding, editing and moving the singularities. This system also lacks control over vector field topology since the vector field created this way may have unspecified singularities.

Theisel [90] proposes a 2D vector field design system in which the user has the complete control over vector field topology. To do so, the user specifies the *topological skeleton* of

the desired vector field and the system creates a piecewise-linear vector field to match the skeleton. However, it can be cumbersome to specify the skeleton for a complicated vector field.

Both of the above topology-based design systems [75, 90] require a planar parameterization, and they cannot be generalized to work for curved surfaces in an obvious way.

All of the existing systems that I review in this chapter have certain traits that I wish to incorporate into the vector field design system. In fact, I borrow techniques from existing systems to at various stages. This will become clear in Chapters 8, 9, and 10. Since I provide several operations in the vector field design system for controlling the vector field topology, it is worth reviewing the most relevant work in vector field topology by the Scientific Visualization community.

6.3 Vector Field Topology

In their well-known work, Helman and Hesselink [37] visualize a vector field by extracting and visualizing its topological skeleton, which consists of the singularities and their connectivity. They propose an efficient method for extracting the topological skeleton for continuous vector fields defined on either a plane or a volume. Their work has inspired a great deal of interests in understanding and visualizing vector fields through topological analysis.

Scheuermann et al. [81] use Clifford algebra to study the non-linear singularities of a vector field and propose an efficient algorithm for merging nearby linear singularities into a higher-order singularity.

Later, Polthier and Preu [69] use Hodge-Decomposition to locate different types of singularities, such as sources, sinks, and centers, in a vector field.

Wischgoll and Scheuermann [100] propose an efficient algorithm for computing periodic orbits in a flow. This is achieved by starting from saddles and follow the flow either forward or backward until it converges onto a path that passes through an element in the mesh for many times.

6.4 Vector Field Simplification

Vector field simplification has been well researched by the Scientific Visualization community. Most of the data sets that come from scientific simulation are difficult to analyze due to the noises in the data. *Vector field simplification* refers to reducing the complexity of a vector field while maintaining its major features. A vector field simplification technique can be either topology-based (TO) or non-topology-based (NTO).

NTO methods perform smoothing to a vector field, either globally or locally. Existing NTO techniques, such as [99, 91], are often based on performing Laplacian smoothing on the potential of a vector field, which is a scalar field. Tong et al. [91] decompose a vector field into three components: curl-free, divergence-free, and harmonic. Smoothing is performed on each component before their results are summed. Smoothing operations reduce vector field complexity and most likely remove a large percentage of the singularities in the original vector field.

TO methods simplifies the topology of a vector field directly. According to Poincaré-Hopf theorem, it is possible to eliminate a pair of singularities with opposite Poincaré indices at the same time. This idea has been formulated into an operation called singularity pair cancelation, which forms the foundation for many existing TO methods. A class of TO methods, such as the work by Edelsbrunner et al. [19, 20] and by Bremer et al. [8], perform pair cancelation on Morse-Smale scalar fields defined on surfaces by changing the values of the scalar function near the singularity pair. This is equivalent to simplifying the gradient vector field of the scalar function. Ni et al. [63] allow the user to design fair Morse functions over a mesh surface, which is equivalent to designing fair gradient vector fields. In their work, the user specifies desired number and configuration of the critical points of the function, and the system performs a multigrid relaxation to determine a Morse function that meets the requirements.

Another class of TO methods perform cancelation on a vector field directly, such as the technique by Tricoche et al. [92]. This technique first locates a region surrounding the singularity pair, and then performs a non-linear optimization on the vector values at the interior vertices such that the Poincaré indices are zero for every triangle inside the region.

My system provides both a NTO method (Section 9.2) and a TO method (Section 9.3.1). The implementations of these methods are rather different from existing techniques. For instance, the singularity pair cancelation method that I implemented in the system is based on Conley index theory. Furthermore, existing topological analysis and simplification techniques are limited to planar and volume domains because it is not clear how to represent a continuous vector field on a mesh surface. I present a piecewise interpolating scheme in Chapter 10 that overcomes this problem, and therefore allows the analysis and editing operations to be adapted to meshes.

CHAPTER VII

BACKGROUND ON SURFACE VECTOR FIELDS

In this chapter, I review some basic facts about vector fields. While the terms in this chapter are defined for surfaces, most of them generalize to higher-dimensions.

Definition 7.0.1. *A* **vector field** V *for a manifold surface* $\mathbf{S}$ *is a smooth vector-valued function that associates to every point* $\mathbf{p} \in \mathbf{S}$ *a tangent vector* $V(\mathbf{p})$.

A vector field defines a system of differential equations:

$$\frac{d\mathbf{p}}{dt} = V(\mathbf{p}). \tag{32}$$

With appropriate restrictions on V, for each point $\mathbf{p}_0 \in \mathbf{S}$, there exists a solution $\mathbf{p} : \mathbb{R} \to \mathbf{S}$ with the property that $\mathbf{p}(0) = \mathbf{p}_0$ ([33, 41]). Because we will be interested in studying multiple solutions simultaneously, it is useful to introduce the notion of the *flow* induced by V that is a continuous function $\varphi : \mathbb{R} \times \mathbf{S} \to \mathbf{S}$ with the property that $\varphi(t, \mathbf{p}_0) = \mathbf{p}(t)$.

The set $\{\mathbf{p}(t) \mid t \in \mathbb{R}\} = \varphi(\mathbb{R}, \mathbf{p}_0)$ is called the *trajectory* through $\mathbf{p}_0$. Uniqueness of solutions to ordinary differential equations guarantees that the set of trajectories form an equivalence relationship on $\mathbf{S}$. In particular, if $\mathbf{q}_0$ belongs to the trajectory of $\mathbf{p}_0$, then $\mathbf{p}_0$ belongs to the trajectory of $\mathbf{q}_0$. This implies that $\mathbf{S}$ can be decomposed into the set of all trajectories. Some trajectories are of particular significance.

Definition 7.0.2. *A point* $\mathbf{p}_0$ *is a* **singularity** *of a vector field* V *if and only if* $V(\mathbf{p}_0) = 0$. *Otherwise,* $\mathbf{p}_0$ *is* **regular**.

A singularity is often known as a *zero*, *equilibrium*, *critical point*, *fixed point*, or *stationary point*. Observe that the trajectory through a singularity $\mathbf{p}_0$ consists of a single point. There are a variety of levels by which a singularity can be classified. The most fundamental has

to do with its stability properties. For this we need the following concept. Let $\mathbf{U} \subset \mathbf{S}$. The *alpha* and *omega* limit sets of $\mathbf{U}$ are defined by

$$\alpha(\mathbf{U}) := \bigcap_{t<0} \mathrm{cl}(\varphi((-\infty, t), \mathbf{U})) \quad \text{and} \quad \omega(\mathbf{U}) := \bigcap_{t>0} \mathrm{cl}(\varphi((t, +\infty), \mathbf{U}))$$

where $\mathrm{cl}(\mathbf{A})$ denotes the closure of the set $\mathbf{A}$. A singularity $\mathbf{p_0}$ is an *attracting singularity* if there exists a neighborhood $\mathbf{U}$ of $\mathbf{p_0}$ such that $\omega(\mathbf{U}) = \mathbf{p_0}$. Similarly, a singularity $\mathbf{p_0}$ is a *repelling singularity* if there exists a neighborhood $\mathbf{U}$ of $\mathbf{p_0}$ such that $\alpha(\mathbf{U}) = \mathbf{p_0}$.

For many of the calculations we will want to use a finer classification based on the *local linearization* of the vector field. For simplicity, let V be a vector field defined for some planar domain $D \subset \mathbb{R}^2 = \{(x, y) \mid x, y \in \mathbb{R}\}$ as follows:

$$V(\mathbf{p}) = \begin{pmatrix} F(x, y) \\ G(x, y) \end{pmatrix}$$

DV, the *Jacobian* of V is defined as:

$$\begin{pmatrix} \frac{\partial F}{\partial x} & \frac{\partial F}{\partial y} \\ \frac{\partial G}{\partial x} & \frac{\partial G}{\partial y} \end{pmatrix} \tag{33}$$

Definition 7.0.3. *For a vector field V defined on $D \in \mathbb{R}^2$, the* **local linearization** *of V for a point $\mathbf{p}_0 \in D$ is given by:*

$$V^*(\mathbf{p}) = V(\mathbf{p_0}) + DV(\mathbf{p}_0)(\mathbf{p} - \mathbf{p}_0) \tag{34}$$

A singularity $\mathbf{p}_0$ is *non-degenerate* if $DV(\mathbf{p}_0)$ has a full rank. A non-degenerate singularity is also known as being a *first-order* or *linear* singularity. For the remainder of this discussion I will assume that $\mathbf{p}_0$ is a non-degenerate singularity.

Let α_1 and α_2 be the eigenvalues of $DV(\mathbf{p}_0)$. Results from linear algebra tell us that α_1 and α_2 are either both real numbers or a pair of conjugate complex numbers. When they are both real numbers, $\mathbf{p}_0$ is one the following:

1. *source*: when $\alpha_1 > 0, \alpha_2 > 0$. This is a repelling singularity for the flow.

2. *sink*: when $\alpha_1 < 0, \alpha_2 < 0$. This is an attracting singularity for the flow.

3. *saddle*: when $\alpha_1\alpha_2 < 0$. $\mathbf{p}_0$ has two incoming trajectories and two outgoing trajectories.

In the case of a pair of conjugate complex numbers, $\mathbf{p}_0$ is one the following:

1. *repelling focus*: when the real parts of α_1, α_2 are positive. $\mathbf{p}_0$ acts much like a source except the local flow leaves in a spiral fashion.

2. *attracting focus*: when the real parts of α_1, α_2 are negative. $\mathbf{p}_0$ acts much like a sink except the local flow moves toward the singularity in a spiral fashion.

3. *center*: when the real parts of α_1, α_2 are zero. If V is a linear vector field, then the nearby trajectories of $\mathbf{p}_0$ are periodic orbits. In the more general case where V is nonlinear the behavior in the neighborhood of $\mathbf{p}_0$ is determined by the higher order terms.

Remark 7.0.4. *Sources and repelling foci are repelling singularities. Sinks and attracting foci are attracting singularities. Centers and saddles are* neither *repelling* nor *attracting singularities.*

Other trajectories of particular importance are *periodic orbits* and *separatrices.*

Definition 7.0.5. *A* **periodic orbit** Γ *is a trajectory* $\mathbf{p}(t)$ *in* D *such that there is a positive number* t_0 *and* $\mathbf{p}(t + t_0) = \mathbf{p}(t)$ *for any* $t \in \mathbb{R}$. *The minimal positive value for* t_0 *is the* **period** *of* Γ. *Furthermore, if there exists a neighborhood* $\mathbf{U}$ *of* Γ *such that* $\omega(U) = \Gamma$ *(*$\alpha(U) = \Gamma$*), then it is an* **attracting (repelling) limit cycle**.

Γ is a loop that contains no singularities. Starting from any point $\mathbf{p}_0 \in \Gamma$ on the orbit and following the vector field, one will get back to $\mathbf{p}_0$ in a finite amount of time. The time between one leaving $\mathbf{p}_0$ and coming back to $\mathbf{p}_0$ is the period. In fact, the period is the same regardless of the starting position. A trajectory near an attracting limit cycle Γ approaches Γ in a spiral fashion without ever reaching it. Similarly, A trajectory near a repelling limit cycle leaves in a spiral fashion.

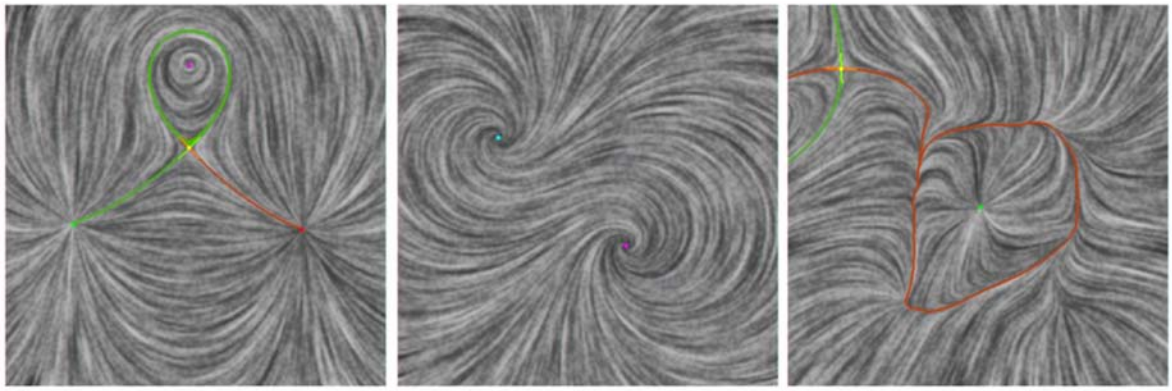

Figure 23: This figure shows different kinds of special trajectories: singularities, separatrices, and limit cycles. In all the images, the singularities are depicted as colored dots and the principle directions for the saddles are drawn as crosses. Furthermore, the incoming separatrices for saddles are shown in green while the outgoing separatrices are shown in red. The vector field in the right contains a limit cycle that separates the flow inside from the flow outside. The visualization technique is based on van Wijk [94].

Definition 7.0.6. *A* **separatrix** Γ *is a trajectory* $\mathbf{p}(t)$ *in* D *such that the limit as* $\lim_{t\to+\infty}\mathbf{p}(t)$ *or* $\lim_{t\to-\infty}\mathbf{p}(t)$ *is a saddle.* Γ *is* **homoclinic** *if and only if* $\lim_{t\to+\infty}\mathbf{p}(t)$=$\lim_{t\to-\infty}\mathbf{p}(t)$. *Otherwise it is* **heteroclinic**.

Basically, a heteroclinic separatrix connects a saddle with another singularity, typically an attractor or a repeller, or a periodic orbit. On the other hand, a homoclinic separatrix leaves a saddle $\mathbf{p}_0$ from one of its outgoing directions and comes back to $\mathbf{p}_0$ in one of its incoming directions. I will describe both types of separatrices more in detail in Section 7.1.

For planar vector fields, the *vector field topology* consists of singularities, separatrices, and periodic orbits. Figure 23 illustrates these special trajectories with three vector fields. Singularities are depicted as colored dots: sources (green), sinks (red), centers (cyan), and saddles (yellow). Repelling and attracting foci are colored in cyan and magenta. Furthermore, incoming and outgoing separatrices are colored in green and red, respectively. The vector field in the right contains an attracting limit cycle. The vector field visualization technique is based on [94].

A vector field is often described either analytically or topologically. The actual analysis is application-dependent. I will review the relevant information for either approach.

7.1 Analytic Descriptions

Two useful analytic characterizations of a vector field are its curl and divergence. The divergence measures the difference between the amount of flow that leaves and that approaches the measurement point. For instance, a source has a positive divergence and a sink has a negative divergence. The curl measures the amount of flow that circles around the measurement point.

Formally, let $\mathbf{p}_0 \in D$ be the measurement point. Furthermore, let $R_\varepsilon = \{\mathbf{p} \mid |\mathbf{p}-\mathbf{p}_0| \leq \varepsilon\}$ and $\gamma_\varepsilon = \partial R_\varepsilon$. R_ε are a family of concentric disks with the center at $\mathbf{p}_0$. Let $\overrightarrow{n_\varepsilon}$ and $\overrightarrow{d_\varepsilon}$ be the outward normal and forward tangent fields with respect to $\mathbf{p}_0$, i.e.,

$$\overrightarrow{n_\varepsilon}(\mathbf{p}) = \frac{\mathbf{p}-\mathbf{p}_0}{|\mathbf{p}-\mathbf{p}_0|} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} \tag{35}$$

$$\overrightarrow{d_\varepsilon}(\mathbf{p}) = \begin{pmatrix} -n_y \\ n_x \end{pmatrix} \tag{36}$$

Then the curl and divergence of V at $\mathbf{p}_0$ are defined as:

$$curl(V)\mid_{\mathbf{p}_0} := \lim_{\epsilon \to 0} \frac{1}{2\pi\epsilon} \oint_{\gamma_\epsilon} V \cdot \overrightarrow{n}\, d\gamma \tag{37}$$

$$div(V)\mid_{\mathbf{p}_0} := \lim_{\epsilon \to 0} \frac{1}{2\pi\epsilon} \oint_{\gamma_\epsilon} V \cdot \overrightarrow{d}\, d\gamma \tag{38}$$

Given a vector field as follows,

$$V(\mathbf{p}) = \begin{pmatrix} F(x,y) \\ G(x,y) \end{pmatrix}$$

The curl and divergence are:

$$curl(V) = \frac{\partial G}{\partial x} - \frac{\partial F}{\partial y} \tag{39}$$

$$div(V) = \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y}. \tag{40}$$

The distributions of the curl and divergence in the domain can help us understand the geometric structure of the trajectories. The two extreme cases are *curl-free* vector fields

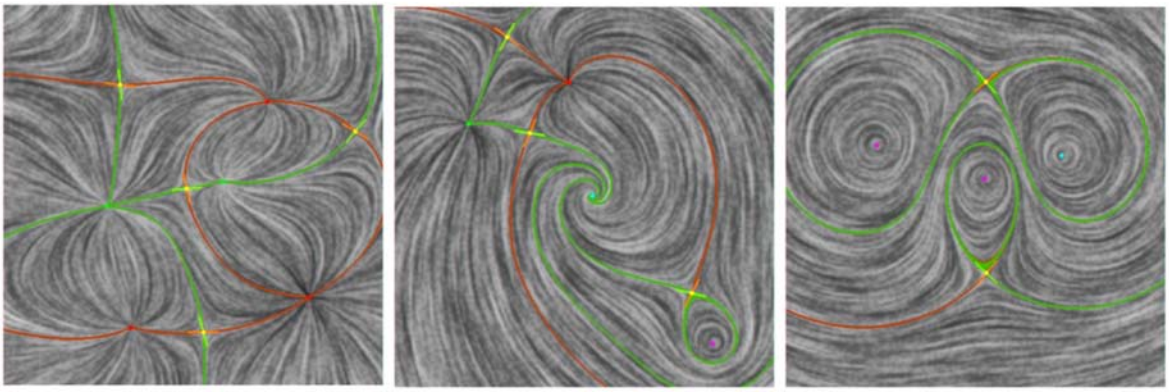

Figure 24: This figure illustrates the main geometric characteristics of different types of vector fields. The vector fields are curl-free (left), divergence-free (right), or generic (middle). Notice the vector field shown in the middle is locally curl-free (upper-left corner), divergence-free (lower-right corner), and neither (middle portion).

in which case the curl is zero everywhere, and *divergence-free* vector fields in which case the divergence is zero everywhere. Depending on the application, a curl-free vector field is also known as a *gradient*, *conservative*, or *irrotational* vector field. A divergence-free vector field is also known as being *Hamiltonian*, *solenoidal*, or *incompressible*. It should be noted that a generic vector field is neither curl-free nor divergence-free. Figure 24 illustrates three vector fields with different analytical behaviors. The vector field shown in the left is curl-free. In this case the typical singularities are sources, sinks, and saddles. Furthermore, the typical separatrices are heteroclinic, which divide the domain into a number of combinatorial quadrilaterals called *basins*. The boundary of each basin consists of a source, a sink, and two saddles in between them. Inside each basin, all the trajectories leave the same source and approach the same sink. The vector field in the right is divergence-free, whose typical singularities are centers and saddles. The separatrices are often homoclicnic, which divide the domain into a number of bounded regions. Inside each region is a family of periodic orbits that circle around the same center. The vector field shown in the middle is locally curl-free (upper-left corner), divergence-free (lower-right corner), and neither (middle).

7.2 Topological Descriptions

The vector field design problem requires that the user be able to control the trajectories of a vector field both locally, even when the local analysis is degenerate, and globally. To

do this requires the introduction of topological characterizations of vector fields. In this section, I will review a well-known topological descriptor, the *Poincaré index*, and a more general characteristic, the *Conley index*.

7.2.1 The Poincaré Index

A singularity $\mathbf{p}_0$ is *isolated* if there exists a compact neighborhood U of $\mathbf{p}_0$ with the property that $\mathbf{p}_0$ is the unique singularity in the interior of U. An isolated singularity $\mathbf{p}_0$ can be characterized by its *Poincaré index*, which is defined in terms of the *winding number* for the *Gauss map*.

Definition 7.2.1. *Let V be a vector field defined on some planar domain D. Let $D_0 \subset D$ be the zero set for V. The* **Gauss map** *$\alpha : D \setminus D_0 \to S^1$ is defined as:*

$$\alpha(x) = \frac{V(x)}{|V(x)|} \tag{41}$$

For a simple closed curve $\Gamma \subset D \setminus D_0$, the Gauss map α induces a continuous map $\alpha|_\Gamma$. If one travels along Γ in the positive direction once, the image under $\alpha|_\Gamma$ necessarily covers the unit circle an integer number of times counting orientation. This integer, $\kappa(V;\Gamma)$, is the *winding number* of V along Γ.

Definition 7.2.2. *Let V be a vector field defined on D and let $R \subset D$ be such that ∂R contains no singularities. The* **Poincaré index** *of R is defined as $\kappa(V;\partial R)$, the winding number of the Gauss map along ∂R. Without causing ambiguity, the* **Poincaré index** *of R is denoted as $\kappa(V;R)$.*

The Poincaré index of a region R has the following properties:

Remark 7.2.3. *Let R_1, R_2 be compact subsets of the domain D such that $R_1 \bigcap R_2 = \partial R_1 \bigcap \partial R_2$, i.e., the only intersection between R_1 and R_2 is on their boundaries. Let $R = R_1 \bigcup R_2$. Furthermore, assume that a vector field V has no singularities on either ∂R_1 or ∂R_2. Then $\kappa(V;R) = \kappa(V;R_1) + \kappa(V;R_2)$.*

Remark 7.2.4. *$\kappa(V;R) = 0$ if R contains no singularity in its interior. This is true even when R is not simply connected, for instance, when R is a ring.*

Under the assumptions in Remark 7.2.4, the Poincaré index of a ring is zero. To see this, imagine we warp the space such that the ring-shaped region R has the canonical form $\{(x, y) \mid 1 \leq x^2 + y^2 \leq 4\}$. The y-axis divides R into two regions: $R_1 = \{(x, y) \mid 1 \leq x^2 + y^2 \leq 4, y \leq 0\}$, and $R_2 = \{(x, y) \mid 1 \leq x^2 + y^2 \leq 4, y \geq 0\}$. Both R_1 and R_2 are simply connected, and their Poincaré indices are zero. Also, $R_1 \bigcap R_2 = \partial R_1 \bigcap \partial R_2$. Therefore, the Poincaré index for R is zero according to Remark 7.2.3.

We are now ready to define the Poincaré index for a point $\mathbf{p}_0 \in D$, possibly an isolated singularity. Let $R_\varepsilon = \{\mathbf{p} \mid |\mathbf{p} - \mathbf{p}_0| \leq \varepsilon\}$ contain no singularities on their boundaries for any $0 < \varepsilon < \varepsilon_0$. Then Remark 7.2.4 implies that $\kappa(V; R_{\varepsilon_1}) = \kappa(V; R_{\varepsilon_2})$ for any $0 < \varepsilon_1, \varepsilon_2 < \varepsilon_0$. Therefore, $\lim_{\varepsilon \to 0} \kappa(V; R_\varepsilon)$ exists. We define $\kappa(V; \mathbf{p}_0)$, the Poincaré index for $\mathbf{p}_0$ as this limit. When $\mathbf{p}_0$ is regular, $\kappa(V; \mathbf{p}_0) = 0$.

For isolated singularities, we have the following results:

1. $\kappa(V; \mathbf{p}_0) = 1$ if $\mathbf{p}_0$ is one of the following: source, sink, center, focus.

2. $\kappa(V; \mathbf{p}_0) = -1$ if $\mathbf{p}_0$ is a saddle.

Let V be a vector field defined on D and let $R \subset D$ be such that ∂R contains no singularities. Furthermore, assume V only contains isolated singularities $\mathbf{p}_1, ..., \mathbf{p}_n$. Then the Poincaré index of R, $\kappa(V; \partial R)$, is equal to the sum of the Poincaré indices of the singularities. If V has degenerate singularities inside R, we can perturb the vector field inside R such that only isolated singularities exist after the perturbation.

The *Poincaré-Hopf theorem* links the vector field topology to the topology of the underlying domain in the following way.

Theorem 7.2.5. *Let S be a closed orientable manifold with Euler characteristic $\chi(S)$. Furthermore, let V be a continuous vector field defined on S with only isolated singularities $\mathbf{p}_1, ..., \mathbf{p}_n$. Then*

$$\sum_{i=1}^{n} \kappa(V; \mathbf{p}_i) = \chi(S) \tag{42}$$

Compare this to Equation 6. Note that the topology of a surface has an impact on the topological characteristics of the scalar fields and vector fields defined on the surface.

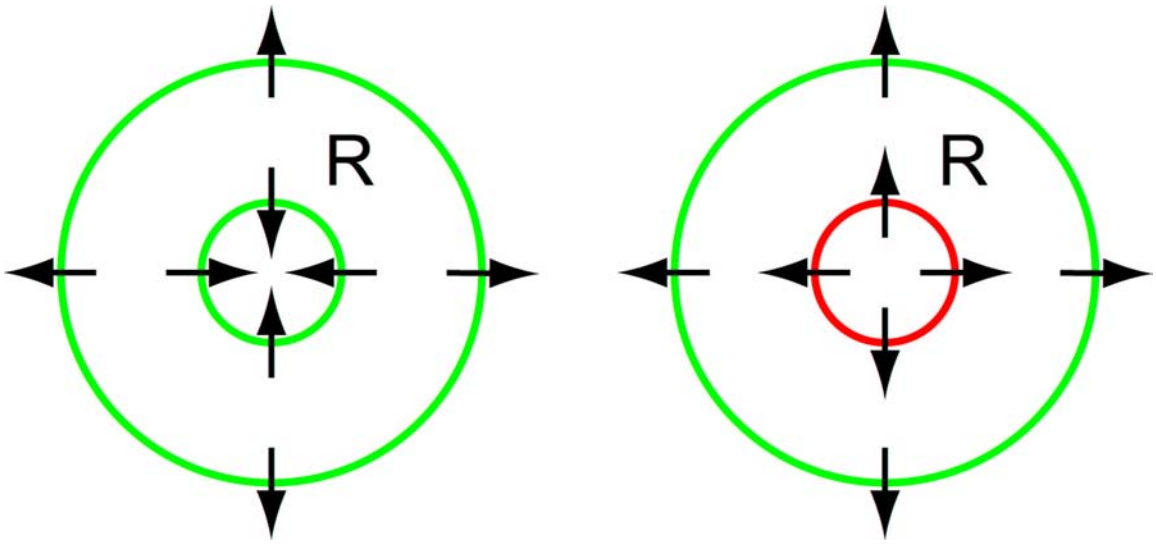

Figure 25: The Conley index is more general than the Poincaré index in that it provides information about periodic orbits. In this figure, the Poincaré indices of the ring-shaped region R are zero for both vector fields. When there are no singularities inside R, the Conley index reveals the vector field in the left necessarily has an repelling limit cycle while the vector field in the right does not.

An immediate corollary of the Poincaré-Hopf theorem is that given a particular vector field V, if one wants to remove a singularity of a positive or negative Poincaré index then one must simultaneously remove a singularity of a Poincaré index with the opposite sign. To perform these removals I borrow basic ideas from Conley index theory (see [13, 61, 62] for further details and references), which is a topological generalization of Morse theory.

7.2.2 The Conley Index

The qualitative structure of a vector field on a planar domain is determined by the number of singularities, the number of periodic orbits, and the set of separatrices. The design of vector fields requires the imposition of additional quantitative information including the location of the singularities, limit cycles and separatrices, and the control of the smoothness and/or curvature of the vector field. The Poincaré index provides a powerful tool for the analysis of singularities. It does not, however, distinguish between a source and a sink, and it is incapable of providing information concerning limit cycles.

In this section, I will review a more general and powerful topological descriptor, the Conley index. The Conley index is defined in the context of arbitrary vector fields that

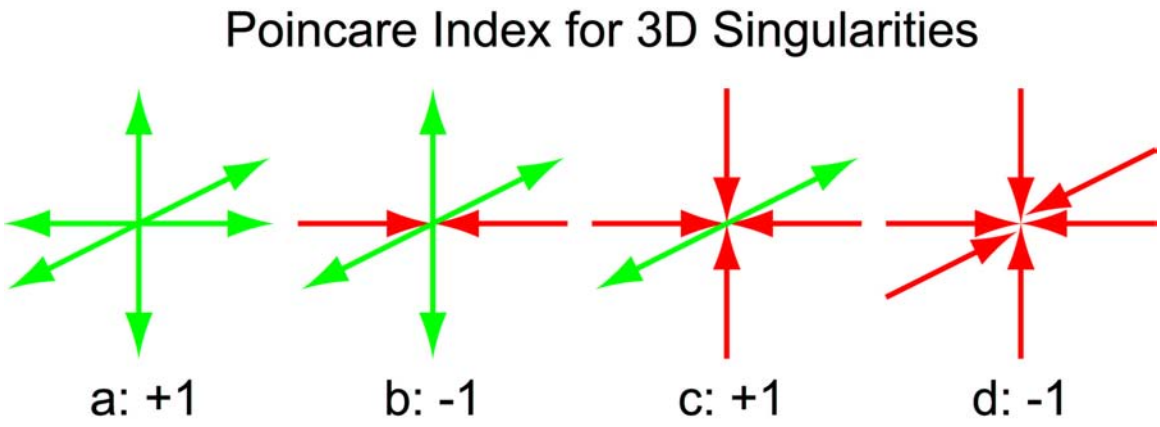

Figure 26: In higher-dimensions, the Poincaré index is not sufficient for determining whether two singularities can cancel. For instance, a source and a sink of a 3D vector field cannot cancel even though they have the opposite Poincaré indices. On the other hand, the Conley index provides the sufficient conditions on whether two singularities can cancel for higher-dimensions.

produce continuous flows. It possesses the continuation properties of the Poincaré index while being able to distinguish between sinks, saddles, and sources. Furthermore, it can be used to identify limit cycles and separatrices. In Figure 25, two vector fields are defined on a ring-shaped region R. Assuming there are no singularities inside R in either case, the Conley index reveals the existence of an repelling limit cycle in R for the vector field in the left. The Poincaré index, on the other hand, does not provide such information. Furthermore, for vector fields defined on higher-dimensional manifolds, it is no longer sufficient for a pair of singularities to cancel when they have opposite Poincaré indices, such as a source and a sink in 3D (Figure 26). On the other hand, the Conley index provides sufficient conditions on whether two singularities cancel for vector fields defined in higher dimensions.

The following concept is the starting point for Conley index theory. Given a region $N \subset \mathbf{S}$, let ∂N denote the boundary of N. A compact set N is an *isolating neighborhood* if for every $\mathbf{p} \in \partial N$, $\varphi(\mathbb{R}, \mathbf{p}) \not\subset N$. This means that the trajectory of any point on the boundary of N leaves either in forward or backward time. The set of boundary points which leave or enter N immediately can be characterized, respectively, by

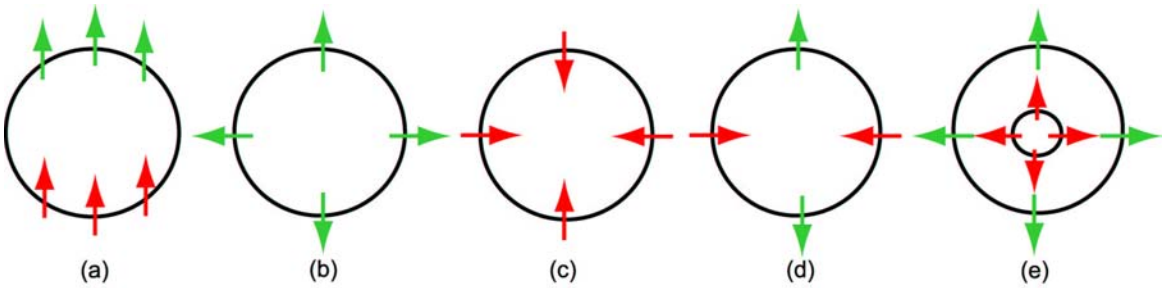

Figure 27: This figure shows five basic scenarios of isolating blocks. Case (a), (b) and (e) are of particular interest since they are used when performing topological editing operations (Section 9.3.1 and 9.3.2).

$$N^- := \{\mathbf{p} \in \partial N \mid \varphi([0,t),\mathbf{p}) \not\subset N,\ \forall t > 0\}$$
$$N^+ := \{\mathbf{p} \in \partial N \mid \varphi((t,0],\mathbf{p}) \not\subset N,\ \forall t < 0\}.$$

A compact set N is an *isolating block* if for each boundary point, there is either a forward or backward trajectory that immediately leaves the region: that is, $\partial N = N^- \cup N^+$. Observe that an isolating block is a special case of an isolating neighborhood.

Given an isolating block N for a vector field V, its Conley index is defined to be the relative homology [44] of N with respect to N^-, i.e. $CH_*(N) := H_*(N, N^-)$. For the purposes of this thesis the computation of this index is fairly simple since the isolating block N will always take the form of a polygonal region and N^- will be a finite number of disjoint sets consisting of boundary edges of N. Idealized isolating blocks and their associated Conley indices are indicated in Figure 27. Recall that the Conley index is a finer invariant than the Poincaré index. In particular, the Conley index distinguishes between isolated sources and sinks. Figure 27 shows five scenarios of isolating blocks and the flow along their boundaries. Their Conley indices are as follows:

$$\text{case (a)} \qquad CH_*(N) = 0$$

$$\text{case (b)} \quad CH_*(N) = \begin{cases} \mathbb{Z} & \text{if } k = 2 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{case (c)} \quad CH_*(N) = \begin{cases} \mathbb{Z} & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{case (d)} \quad CH_*(N) = \begin{cases} \mathbb{Z} & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{case (e)} \qquad CH_*(N) = 0$$

Case (a) and (e) have the trivial Conley index, and (b), (c), and (d) have the Conley index of a source, a sink, and a saddle, respectively. Of particular interest are case (a), (b), and (e). I construct regions of these types for topological editing operations (Section 9.3.1 and 9.3.2). Starting from next chapter, I will describe my vector field design systems for both planar domains and mesh surfaces in detail.

CHAPTER VIII

VECTOR FIELD DESIGN FOR PLANAR DOMAINS

In this chapter, I describe my vector field design system for planar domains. The system consists of three stages:

1. Initialization: The user quickly creates a vector field with a small set of constraints. At this stage, vector field topology is not a concern.
2. Analysis: The system performs both geometric and topological analysis on the current vector field and provides visual feedback to the user.
3. Editing: The user modifies the vector field through a set of pre-defined editing operations.

The user may perform many editing operations before accepting the result. The initialization and analysis stages are relatively straightforward, and I will describe them in Section 8.1 and 8.2, respectively. The editing stage is at the core of my vector field design system, which I will describe in Chapter 9.

8.1 Creating Initial Vector Fields

The first stage allows a user to easily create an initial vector field without being concerned with vector field topology. There are two ways of creating such a field in previous work: relaxation [93, 97], and using basis vector fields [70, 94]. I adopt van Wijk's approach [94] of using basis vector fields because of its intuitive nature and its simplicity. In this approach, every user-specified constraint is used to create a basis vector field defined in the plane. An initial vector field is then constructed as a weighted sum of these basis vector fields. I will refer to a user-specified constraint as a design element. There are two types of design elements: *singular* and *regular*.

A singular element corresponds to a vector field that has a singularity of a user-specified type at a desired location. For instance, if the user wishes to have an isotropic source at location $\mathbf{p}_0 = (x_0, y_0)$ with strength $k > 0$, the system will create the following vector field for any point $\mathbf{p} = (x, y)$ in the plane as:

$$V(\mathbf{p}) = e^{-d\|\mathbf{p}-\mathbf{p}_0\|^2} \begin{pmatrix} k & 0 \\ 0 & k \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \tag{43}$$

Here, d is a decay constant that is used to control the amount of influence of the basis vector field. Other isotropic singular elements include a sink, a saddle, a counter-clockwise center, and a clockwise center, whose matrices are the following:

$$\begin{pmatrix} -k & 0 \\ 0 & -k \end{pmatrix}, \quad \begin{pmatrix} k & 0 \\ 0 & -k \end{pmatrix}, \quad \begin{pmatrix} 0 & -k \\ k & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & k \\ -k & 0 \end{pmatrix}. \tag{44}$$

The system allows the user to modify the scale, the orientation and the center location of each singular element as well as to remove an existing singular element. Modifications to singular elements will result in more complicated matrices. For example, an anisotropic source with eigenvalues k_1 and k_2 and eigenvectors v_1 and v_2 will result in the following matrix:

$$\begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix} \begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix} \begin{pmatrix} v_1 & v_2 \end{pmatrix} \tag{45}$$

A regular element refers to a vector field that has a particular nonzero vector value V_0 at a desired location $\mathbf{p}_0$. Again, the system creates a basis vector field as follows:

$$V(\mathbf{p}) = e^{-d\|\mathbf{p}-\mathbf{p}_0\|^2} V_0 \tag{46}$$

The resulting vector field is interactively updated and displayed as the user continues to make adjustment to the set of regular and singular elements. Figure 28 shows two vector fields that were generated using singular elements (left, highlighted by the colored boxes) and regular elements (right, highlighted by the colored arrows). In practice, both types of specifications can be combined to create an initial vector field. Notice that summing the

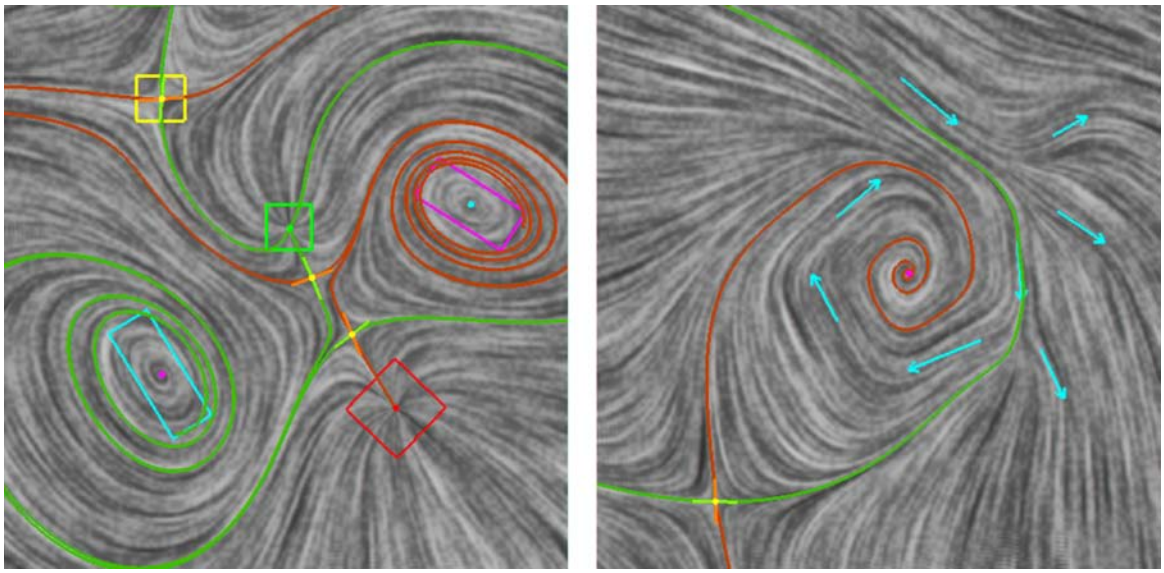

Figure 28: This figure shows two types of user-specified constraints for creating the initial vector field with *singular elements* (left, highlighted by the colored boxes), and *regular elements* (right, highlighted by the colored arrows). The centers of the colored boxes are the locations of the desired singularities. Notice in both cases, there are singularities not specified by the user.

basis vector fields may cause unwanted singularities to appear, i.e., the ones that are not at the centers of any colored box in this figure. The unwanted singularities will be handled through the topological editing operations that I will describe in Chapter 9.

8.2 Analysis

The initial vector field created in the first stage is difficult to analyze because of its complicated formula. Furthermore, analytical formulas are in general not available for mesh surfaces, which often lack a global parameterization. To perform analysis in a fast and efficient manner and to be able to generalize the method to surfaces, I use a piecewise approximation in which the underlying domain is tessellated by a triangular mesh. The vector values are sampled at the vertices according to the analytic formula and are linearly interpolated on the edges and across the interiors of the triangles.

Let a vector field V be given by the set of vectors $\{W_1, W_2, ..., W_n\}$ at the mesh vertices $\{v_1, v_2, ..., v_n\}$. For a point $\mathbf{p}$ inside a triangle $\{A = v_{T_1}, v_{T_2}, v_{T_3}\}$ whose barycentric

coordinates are $(\delta_1, \delta_2, \delta_3)$, we have

$$V(\mathbf{p}) = \sum_{j=1}^{3} \delta_j W_{T_j} \tag{47}$$

or, under some local coordinate system of T,

$$V(\mathbf{p}) = \mathbf{M_T} \begin{pmatrix} x_\mathbf{p} \\ y_\mathbf{p} \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}, \text{ where } \mathbf{M_T} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{48}$$

This representation does not require a global analytical formula, and it is compatible with many graphics applications that make use of vector fields. Furthermore, Equations 47 and 48 can be adapted to represent a surface vector field (Chapter 10).

My vector field design system performs the following analysis for each triangle:

- Compute the divergence and curl.
- Determine the Poincaré index κ.
- If κ is $+1$, determine the location of the singularity inside the triangle.
- If κ is -1, determine the location of the saddle inside the triangle and its outgoing and incoming directions.

Let a triangle $T = \triangle ABC$ have vector values W_A, W_B, and W_C at its vertices. For convenience, I use the following local coordinate system so that the origin coincides with A and the positive X-axis coincides with the vector $\overrightarrow{AB}$. There is a unique right-hand orthonormal system in the plane once the X-axis is chosen. Let X and Y be the unit vectors in the positive X-axis and Y-axis. Then A, B and C have the following coordinates:

$$x_A = 0 \tag{49}$$

$$y_A = 0 \tag{50}$$

$$x_B = |\overrightarrow{AB}| \tag{51}$$

$$y_B = 0 \tag{52}$$

$$x_C = \overrightarrow{AC} \cdot X \tag{53}$$

$$y_C = \overrightarrow{AC} \cdot Y \tag{54}$$

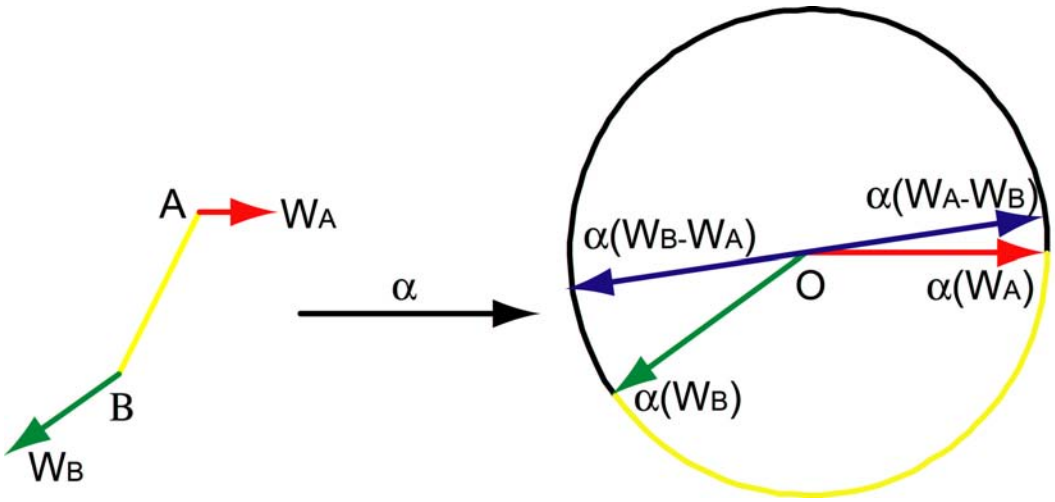

Figure 29: This figure shows the image of the Gauss map α on a linear interpolating vector field on an edge AB. When W_A and W_B are not co-linear, α maps vector field on the edge AB to the shorter arc between $\alpha(W_A) = \frac{W_A}{|W_A|}$ and $\alpha(W_B) = \frac{W_B}{|W_B|}$

then $a, b, ...f$ in Equation 48 are:

$$a = \frac{(W_B - W_A) \cdot X}{x_B} \tag{55}$$

$$b = \frac{(x_B(W_C - W_A) - x_C(W_B - W_A)) \cdot X}{x_B y_C} \tag{56}$$

$$c = \frac{(W_B - W_A) \cdot Y}{x_B} \tag{57}$$

$$d = \frac{(x_B(W_C - W_A) - x_C(W_B - W_A)) \cdot Y}{x_B y_C} \tag{58}$$

$$e = W_A \cdot X \tag{59}$$

$$f = W_A \cdot Y \tag{60}$$

8.2.1 The Curl and Divergence

With Equation 48, the curl and divergence for each triangle are $c - b$ and $a + d$, respectively.

8.2.2 The Poincaré Index

To determine the Poincaré index of a triangle T, let us examine the Gauss map α on $\Gamma = \partial T$, the boundary of the triangle. Assume that there are no singularities on Γ. Then we define $\alpha_{AB} := \alpha(\{(1-t)W_A + tW_B \quad | \quad t \in [0,1]\})$, that is, α_{AB} is the shorter arc on the unit circle between W_A and W_B (Figure 29, the yellow arc in the right image). In fact,

$$\lim_{t \to +\infty} \frac{(1-t)W_A + tW_B}{|(1-t)W_A + tW_B|} = \frac{W_B - W_A}{|W_B - W_A|} \tag{61}$$

$$\lim_{t \to -\infty} \frac{(1-t)W_A + tW_B}{|(1-t)W_A + tW_B|} = \frac{W_A - W_B}{|W_A - W_B|} \tag{62}$$

In other words, the image of a linear vector field along a line under the Gauss map is a semi-circle. The image of the same vector field along any finite line segment only covers part of the semi-circle. Therefore, $|\alpha_{AB}| < \pi$. Similarly, $|\alpha_{BC}| < \pi$ and $|\alpha_{CA}| < \pi$. Since the Poincaré index κ satisfies $2\pi\kappa = \alpha_{AB} + \alpha_{BC} + \alpha_{CA}$, we have $|\kappa| \leq 1$. To calculate the Poincaré index for the triangle $T = \triangle ABC$, I simply examine the sum $\alpha_{AB} + \alpha_{BC} + \alpha_{CA}$, which must be -2π, 0, or 2π. These values correspond to a saddle, no singularity, or singularity with index one (source, sink, center, or focus). Figure 30 shows two examples: the vector field in the top portion has $\kappa = 1$ while the one for the bottom portion has $\kappa = -1$.

8.2.3 Singularities

To determine the location of the singularity inside a triangle T with a none-zero Poincaré index, I solve for

$$\mathbf{M_T} \begin{pmatrix} x_\mathbf{p} \\ y_\mathbf{p} \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = 0 \tag{63}$$

When $\mathbf{M_T}$ has full-rank, there is a unique solution to the system, which is a singularity. However, this singularity is valid for the original vector field only when it is situated inside T. This can be easily verified by computing its barycentric coordinates. However, Equation 63 does not generalize to surface vector fields (Chapter 10). Figure 31 illustrates a different way of computing the location of a singularity if the Poincaré index of the triangle is not zero. Recall that our vector field is piecewise linear in T, which implies that the vector field is also linear on any line segment. First, there is a point D on the edge AC such that $\alpha(W_D) = -\alpha(W_B)$, which we can determine via linear interpolation. Next, on line segment BD there is a unique point E with a zero vector value, which again can be found through

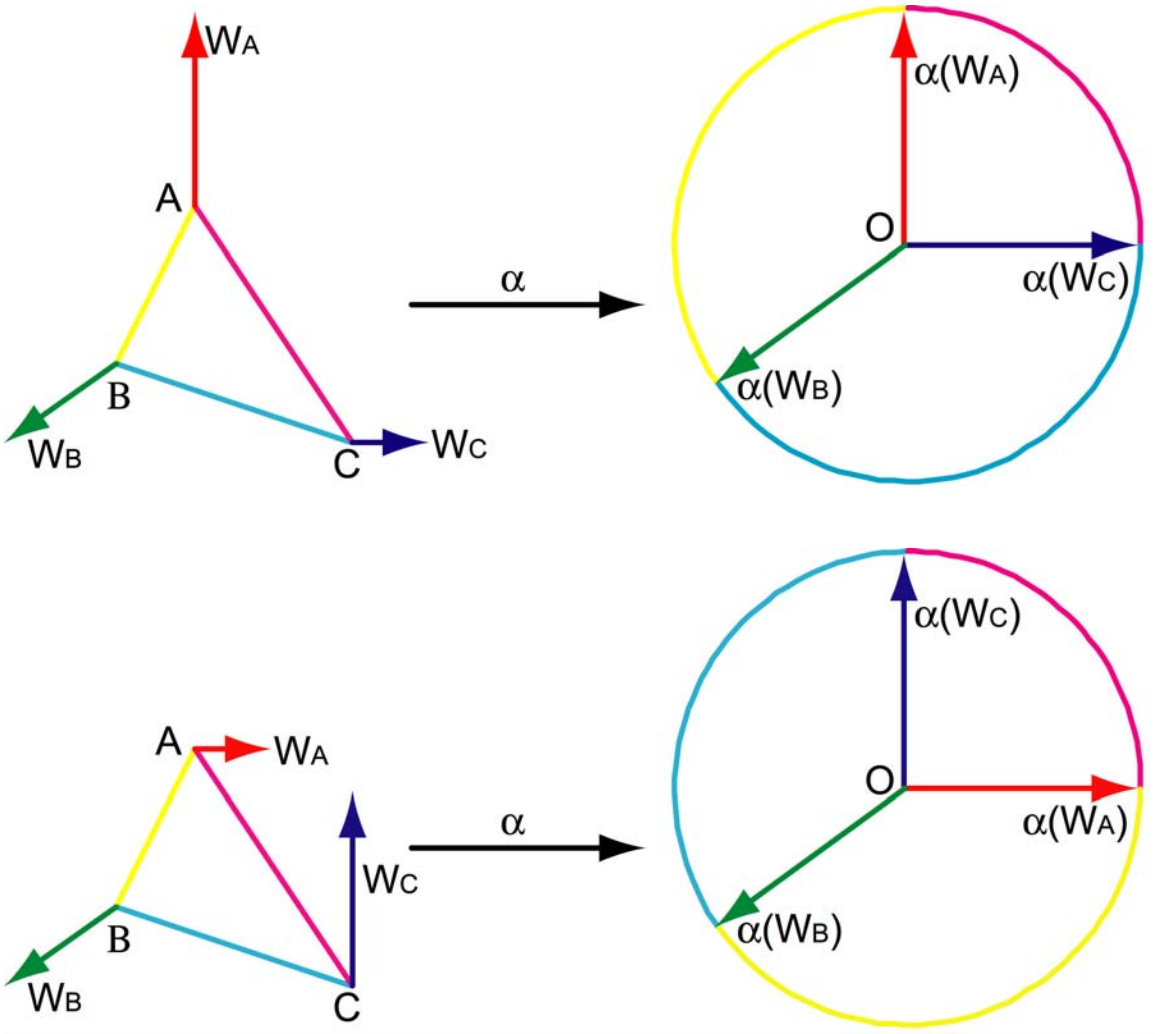

Figure 30: This figure shows the computation of the Gauss map α and the Poincaré index for a triangle $T = \triangle ABC$ with vector values W_A, W_B, and W_C. In the top portion, $\alpha_{AB} + \alpha_{BC} + \alpha_{CA} = 2\pi$ and the Poincaré index $\kappa = 1$. In the bottom portion, $\alpha_{AB} + \alpha_{BC} + \alpha_{CA} = -2\pi$ and $\kappa = -1$.

linear interpolation. This latter method of locating the singularity inside a triangle can be adapted to curved surfaces.

If the singularity is a saddle, I obtain the outgoing and incoming directions by computing the eigenvectors for $\mathbf{M_T}$.

8.2.4 Separatrices

To compute the topological skeleton of the vector field, I follow the approach of Helman and Hesselink [37]. Starting from every saddle point, I follow the flow forward in its outgoing

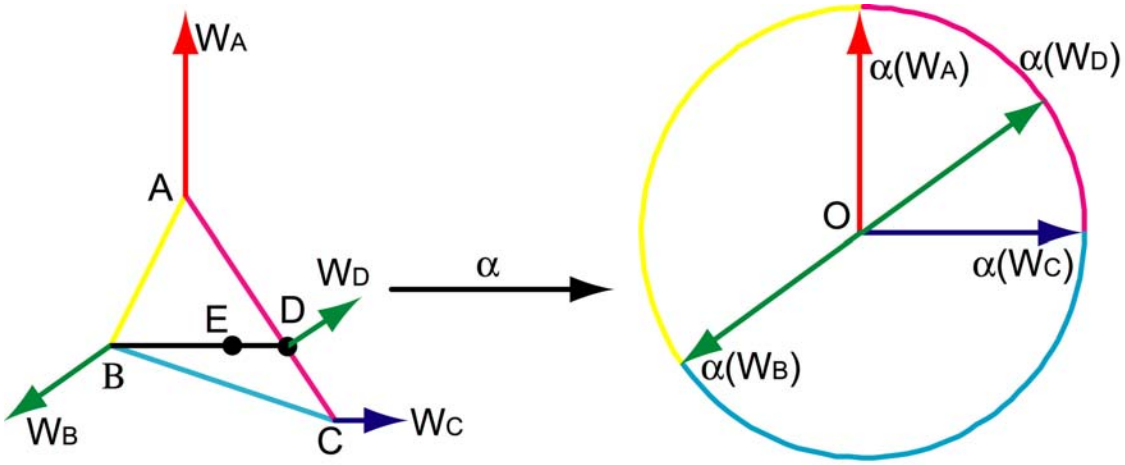

Figure 31: This figure shows how a singularity is found in a triangle, if it exists. First, the unique point $D \in AC$ is found such that the vector value W_D at D is co-linear with W_B but in opposite directions. Then the unique point $E \in BD$ is found whose vector value is zero.

directions until the flow is stopped at a singularity or hits the boundary of the domain. To trace a trajectory away from a saddle I use Runge-Kutta algorithm with adaptive stepsize control [11]. This gives us the two outgoing separatrices. Similarly, I obtain the two incoming separatrices by following the flow backward along the incoming directions of the saddle. Figure 24 and 28 show the topological skeletons of the corresponding vector fields. Notice that this method often causes a homoclinic separatrix to be computed twice, one from the outgoing direction and the other from the incoming direction.

Once the analysis of a vector field is available, the user can start editing the flow.

CHAPTER IX

EDITING

The vector field editing stage is at the heart of my vector field design system. The operations provided at this stage give the user detailed control over the number and location of the singularities as well as the analytic characteristics such as the divergence and curl.

The set of useful of editing operations are application-dependent. In texture synthesis and non-photorealistic rendering, the user may wish to remove unwanted singularities or to move them to less visible regions. Fluid simulation may require adjusting the amount of the curl and divergence in the external force. Furthermore, noisy data sets often contain a large number of singularities and rather complex behaviors. Simplifying the flow while maintaining its major features is a necessary task for any vector field design system. In my system, the following operations are provided.

1. Matrix actions on flows such as: *flow rotations* and *flow reflections*.
2. *Flow smoothing* within a user-defined region.
3. Topological editing operations such as: *singularity pair cancelation* and *singularity movement*.

Matrix actions are used to adjust flow characteristics such as the curl and divergence, as well as to overcome numerical instabilities associated with regions of high curl and regions near saddles. Flow smoothing is an efficient vector field simplification operation that also simplifies vector field topology. Topological editing operations are used to provide direct control over the number and location of the singularities in the vector field.

Since we use a piecewise linear approximation, all the editing operations affect the vector values at the vertices only. These vector values are then extended to a continuous vector field defined over the whole mesh surface through the interpolation scheme described in Section 8.2.

9.1 Matrix Actions on Flows

Any 2×2 matrix M can be considered as a vector field operator, i.e., it acts on a vector field and produces another vector field. Let G be the set of all the 2×2 matrices. Let us consider the following two subsets of G.

$$R = \{ \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \mid \theta \in [0, 2\pi) \} \tag{64}$$

$$F = \{ \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ -\sin(\theta) & -\cos(\theta) \end{pmatrix} \mid \theta \in [0, 2\pi) \} \tag{65}$$

R is the set of all the rotational matrices while F consists of all the reflectional matrices. Actions of the elements in R and F are of particular interests in my system, and I will describe them in detail in the following sections.

9.1.1 Flow Rotation

For any $\theta \in [0, 2\pi)$, R_θ is a vector field operator that acts on a vector field V to produce another vector field $R_\theta(V)$ as follows:

$$(R_\theta(V))(\mathbf{p}) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} V(\mathbf{p}) \tag{66}$$

Basically, R_θ rotates the vector values by θ everywhere. It is straightforward to verify that for any θ,

$$div(R_\theta(V)) = \cos(\theta) div(V) - \sin(\theta) curl(V) \tag{67}$$

$$curl(R_\theta(V)) = \cos(\theta) curl(V) + \sin(\theta) div(V) \tag{68}$$

$$(curl(R_\theta(V)))^2 + (div(R_\theta(V)))^2 = (curl(V))^2 + (div(V))^2 \tag{69}$$

These equations imply that for any vector field V and any point $\mathbf{p}$ in the domain, there exists an appropriate rotation of V such that the new vector field has zero curl at $\mathbf{p}$. Similarly, there is a rotation of V that gives zero divergence at $\mathbf{p}$. More specifically, a

curl-free vector field can be rotated into a divergence-free vector field and vice versa with a 90° rotation.

From a topological point of view, flow rotations do not alter the set of the singularities nor their locations. Furthermore, they do not alter the Poincaré indices of the singularities. Please see Appendix A for the proofs of these statements.

A singularity $\mathbf{p}_0$ with a Poincaré index of $+1$ can be converted into a source with an appropriate rotation of θ. Here, θ is determined such that $R_\theta(V)$ has zero curl and a positive divergence at $\mathbf{p}_0$. Let $\mathcal{C}$ and $\mathcal{D}$ be the curl and divergence of V at $\mathbf{p}_0$, and $\mathcal{C}'$ and $\mathcal{D}'$ be the curl and divergence of $R_\theta(V)$ at $\mathbf{p}_0$. From Equation 68, we have

$$\theta = \arctan \frac{-\mathcal{C}}{\mathcal{D}} \tag{70}$$

There are two θ's in $[0, 2\pi)$ that satisfy this equation. Using Equation 67, we can determine the unique θ such that $\mathcal{D}' > 0$.

A saddle remains a saddle under flow rotations. However, its incoming and outgoing directions are rotated, possibly by different amounts. For an isotropic saddle in which the absolute values of both eigenvalues are equal, the incoming and outgoing directions are rotated by $\theta/2$.

Flow rotations can be used to adjust the divergence and curl in the flow. Moreover, the topological properties of these operations make them essential for the topological editing operations such as singularity pair cancelation (Section 9.3.1) and singularity movement (Section 9.3.2), especially in regions of high curl.

In practice, flow rotation is applied at the vertices only. The piecewise linear representation guarantees that the vector values for any point in the interior of an edge or a triangle is also rotated by the same amount. In the top row of Figure 32, a vector field (upper-left) is rotated by 45° (upper-middle) and 90° (upper-right).

9.1.2 Flow Reflection

For any $\theta \in [0, 2\pi)$, F_θ is a vector field operator that acts on a vector field V to produce a vector field $F_\theta(V)$ as follows:

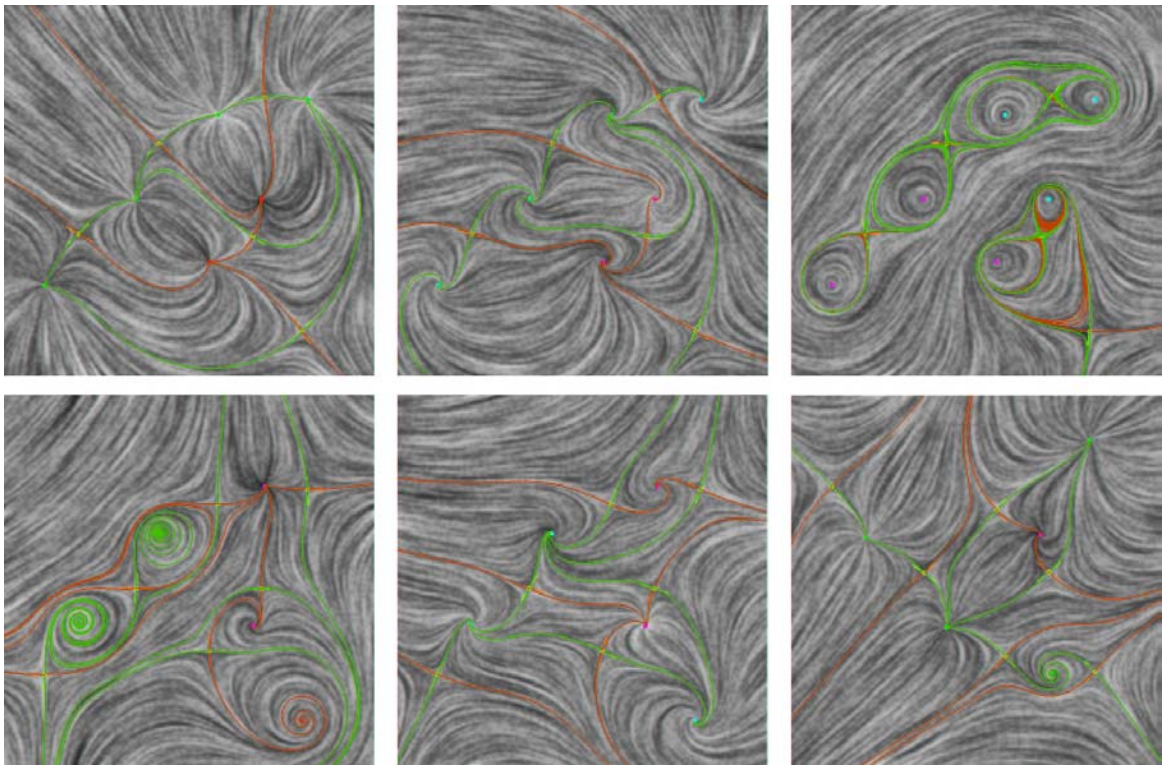

Figure 32: In this figure, a vector field (upper-left) is rotated by 45° (upper-middle) and 90° (upper-right). The vector fields in the lower column are obtained by performing reflections about the Y-axis on the corresponding vector fields in the upper-row. Notice how rotations change the divergence and curl in the vector field. Also, notice flow rotations do not alter the number, the location, and the Poincaré index of the singularities, while flow reflections negate the sign of the Poincaré indices.

$$(F_\theta(V))(\mathbf{p}) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ -\sin(\theta) & -\cos(\theta) \end{pmatrix} V(\mathbf{p}) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} (R_\theta(V))(\mathbf{p}) \tag{71}$$

Basically, F_θ induces a reflection on the vector values with respect to axis $\cos(\frac{\theta}{2})X + \sin(\frac{\theta}{2})Y = 0$ everywhere. This is different from a reflection of the domain itself. Also, it is straightforward to verify that $F_\theta(F_\theta(V)) = V$, i.e., $F^2 = Id$.

For planar domains, flow reflections do not alter the number and location of the singularities in V. On the other hand, they negate the sign of the Poincaré indices for isolated singularities. Please see Appendix A for the proofs of these statements.

Just as flow rotations can convert a singularity with a +1 Poincaré index into a source, flow reflections can turn any saddle into a source with an appropriate choice of the reflection

axis. These operations are important for the singularity movement operation (Section 9.3.2) because they help overcome the numerical difficulties associated with saddles.

Figure 32 shows the effect of applying flow rotations and reflections to a planar vector field. The actions are $R_0 = Id$ (upper-left), $R_{\frac{\pi}{4}}$ (upper-middle), and $R_{\frac{\pi}{2}}$ (upper-right) in the top row, and F_0 (lower-left), $F_{\frac{\pi}{4}}$ (lower-middle), and $F_{\frac{\pi}{2}}$ (lower-right) in the bottom row. Notice that in all instances, the number and location of the singularities do not alter. Flow rotations maintain the Poincaré indices while flow reflections negate their signs.

The concepts of flow rotations and flow reflections are not new. Theisel and Weinkauf [89] define four types of vector field operations for feature-matching between vector fields. The operations that they have defined include rotation and scaling (including reflection). However, I believe that the idea of using flow rotations and reflections for singularity pair cancelation and singularity movement is novel.

9.2 Flow Smoothing

A vector field often contains regions in which it is noisy. The users sometimes wish to simplify the vector field so that its large features become more distinguishable. Vector field simplification has been well researched by the Scientific Visualization community. There are two classes of simplification methods: topology-oriented (*TO*) [92, 19, 20, 8] and non topology-oriented (*NTO*) [99, 91]. TO methods are concerned with maintaining the main topological features, such as the singularities and their connectivity (separatrices) during the simplification process. On the other hand, NTO methods are mainly concerned with maintaining the smoothness of the flow and its behaviors near prominent singularities. As a byproduct, NTO methods can also simplify vector field topology without explicit control. The method I describe next is a NTO method. In Section 9.3.1, a TO method (singularity pair cancelation) will be presented.

The smoothing operation is carried out in two stages. First, the user specifies a simply connected region by drawing a closed loop in the domain. Second, the vector field inside the region is replaced with a "simpler" vector field. The key for this operation is to let the user decide the region for smoothing. Once the region is determined, a number of known

smoothing techniques, such as [99, 91] can be used to replace the flow inside. Here, I use a smoothing approach on the vector values instead of on the vector field potentials as in [91].

Given a vector field V and a user-specified region R, V is replaced inside R with another vector field V' by solving the vector-valued Laplace equation inside R, with V being fixed on the boundary. Let

$$\mathbf{V}'(\mathbf{p}) = \begin{pmatrix} F'(p_1, p_2) \\ G'(p_1, p_2) \end{pmatrix}$$

Then the new vector field V' is given by:

$$\begin{pmatrix} \bigtriangledown^2 F' = 0 \\ \bigtriangledown^2 G' = 0 \end{pmatrix} \tag{72}$$

In practice, the user-specified region R is part of the underlying mesh that is used to represent the planar domain. To solve Equation 72 on this discrete mesh, the vector values are fixed at the boundary vertices of R, i.e.,

$$F' = F, \quad G' = G \tag{73}$$

The vector values of F' and G' for an interior vertex v_i is determined by:

$$\begin{pmatrix} F'(v_i) \\ G'(v_i) \end{pmatrix} = \sum_{j \in J} \omega_{ij} \begin{pmatrix} F'(v_j) \\ G'(v_j) \end{pmatrix} \tag{74}$$

Here, J is the set of index j's such that (v_i, v_j) is an edge in the mesh. The weights ω_{ij}'s are defined according to the mean-value coordinates of Floater [23] since this method guarantees ω_{ij}'s to be non-negative.

Note that the Laplace Equation 72 is vector-valued, in contrast to the more familiar Laplace equation over a scalar value. This operation smooths the flow geometrically and will often reduce the number of singularities of V in R. In Figure 33, a complicated vector field with many singularities (left) is converted into a vector field with only one singularity (right) through smoothing. The boundary of the user-specified region is highlighted with a white loop. Notice that the vector field is not altered outside the user-specified region.

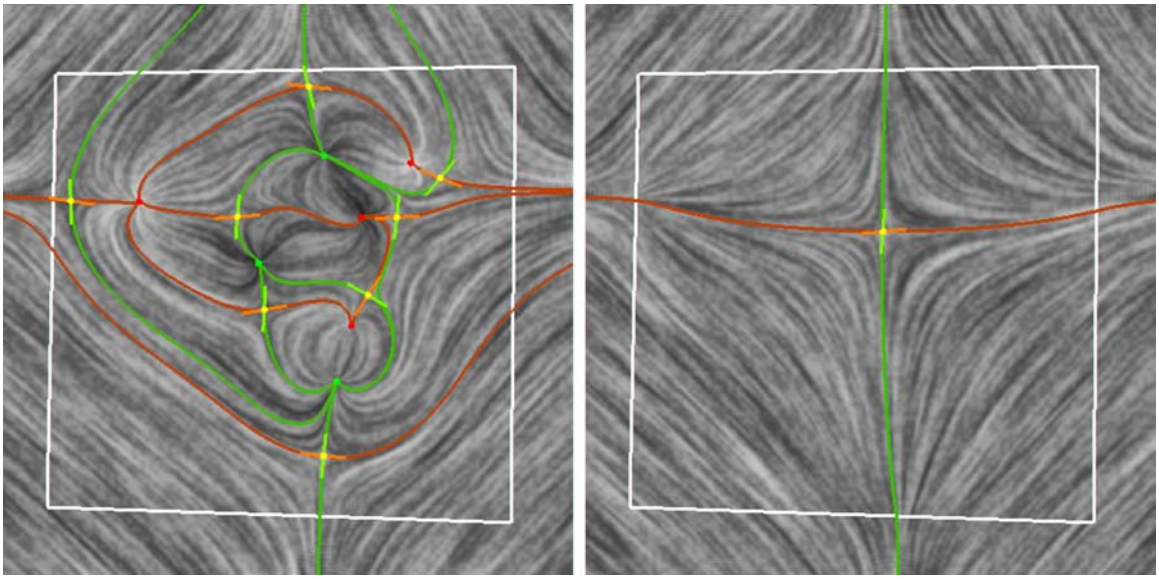

Figure 33: This figure shows the results of applying flow smoothing to a user specified region (inside the white boundary). Notice that the vector field defined outside the region is not changed.

Later, I make use of the flow smoothing operation to perform topological editing operations such as singularity pair cancelation (Section 9.3.1) and singularity movement (Section 9.3.2).

Remark 9.2.1. *On a planar domain, a flow smoothing operation commutes with any matrix action on flows. In particular, it commutes with flow rotations and flow reflections.*

This allows us to change the characteristics of a vector field so that it is easy to process. For example, with proper flow rotations, any singularity pair cancelation can be turned into a source/saddle cancelation. Also, with proper flow rotations and reflections, moving any singularity can be converted into moving a source.

9.3 Topological Editing Operations

A vector field often contains unwanted singularities that will cause visual artifacts in their respective applications. To allow a user to control the number and location of the singularities in a vector field, I describe two topological editing operations: *singularity pair cancelation*, and *singularity movement*. Singularity pair cancelation refers to canceling a pair of singularities with opposite Poincaré indices, and it is used to remove unwanted singularities in the initial vector field. Singularity movement refers to moving a singularity to

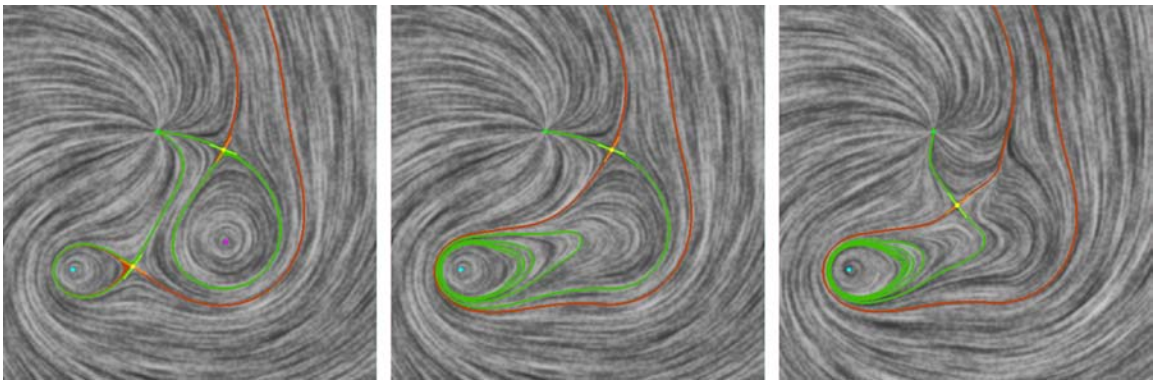

Figure 34: This figure shows the results of applying two successive topological editing operations to a divergence-free vector field (left). First, a pair of center and saddle is canceled (middle). Then, the remaining saddle is moved (right). The success of these operations requires finding appropriate rotations and reflections that change the local characteristics of the vector field near the singularities.

a more desirable location. Both operations provide topological guarantees in that they only affect the singularities that the user wishes to cancel or to move. Figure 34 shows the results of applying two successive topological editing operations to a divergence-free vector field (left). First, a pair of center and saddle is canceled (middle). Next, the remaining saddle is moved (right). The algorithms that I used for both operations are based on Conley index theory.

9.3.1 Singularity Pair Cancelation

The vector field design system provides the capability of eliminating a singularity. As discussed in Section 7.2, singularity elimination must be performed for a pair of singularities with opposite Poincaré indices. This operation is therefore called *singularity pair cancelation.*

There have been several prior pair cancelation methods for simplifying scalar fields on surfaces [19, 20, 8]. These techniques achieve singularity pair cancelation for the gradient field by modifying the scalar values in a nearby region. It is not clear how these techniques can be used for non-gradient vector fields, which need not correspond to any scalar functions.

Tricoche et al. [92] have proposed a pair cancelation technique for vector fields by allowing a saddle to be canceled with either a source or a sink. To achieve this, they first find a narrow neighborhood that encloses the singularity pair and their connecting orbit. They

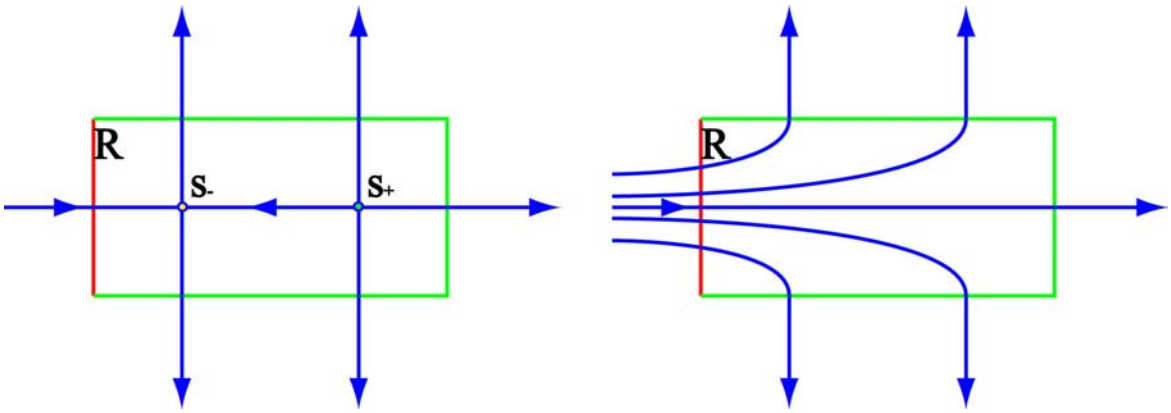

Figure 35: This figure illustrates my two-step algorithm for singularity pair cancelation between a source $\mathbf{s}_+$ and a saddle $\mathbf{s}_-$. In the left portion, a region R is found to enclose both singularities and its boundary consists of two segments: inflow (red) and outflow (green). The vector field inside R is replaced with a flow that has no singularities (right).

then perform an iterative non-linear optimization on the vector values at the interior vertices of this region so that the Poincaré index for every triangle is zero. There are a number of issues with this approach. From a theoretical viewpoint, any simplification technique based on the Poincaré index cannot be applied to the cancelation of a repeller/attractor pair in which one of the entities is a limit cycle (Section 7.2). From a numerical point of view, this technique is not robust in handling pair cancelation that involves a center or a focus with high curl. Furthermore, the non-linear optimization technique is computationally expensive, and it does not guarantee that an acceptable solution can be found.

In this work, I present a new pair cancelation technique that is carried out in two stages. This technique is based on Conley index theory. First, an isolating block R with the trivial Conley index is located such that R encloses the singularity pair in its interior. Second, the flow inside R is replaced with a new, singularity-free vector field. Figure 35 illustrates the idea. A similar two-stage approach also applies when moving a singularity (Section 9.3.2).

Let $\mathbf{s}_+$ and $\mathbf{s}_-$ be the positive-indexed and negative-indexed singularities that the user wishes to cancel. With a proper flow rotation, $\mathbf{s}_+$ can always be converted into a source while $\mathbf{s}_-$ remains a saddle. From now on, let us assume that $\mathbf{s}_+$ is a source.

We will make use of the following definition.

Definition 9.3.1. *For a given vector field V, let φ denote the flow induced by V. For a*

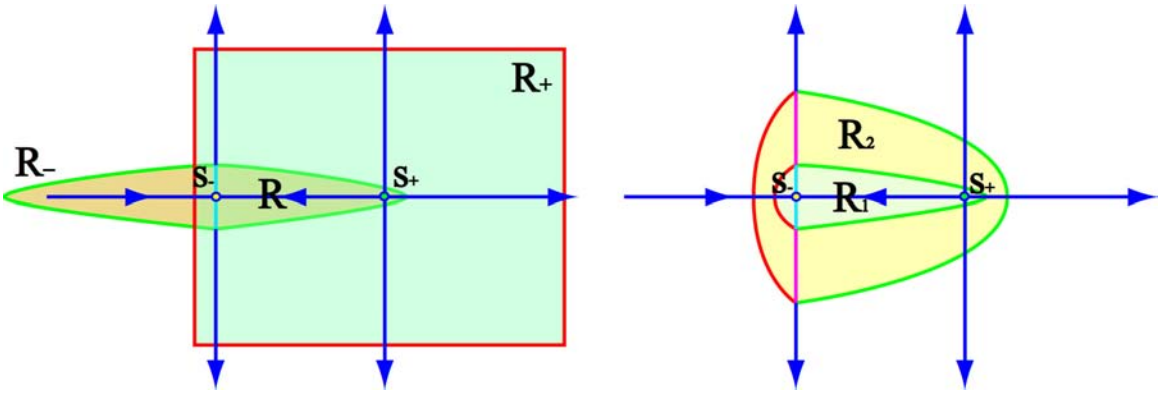

Figure 36: This figure illustrates the construction of an isolating block R for singularity pair cancelation. In the left portion, a region R_+ is generated by following the flow forward from a neighborhood of $\mathbf{s}_+$. Similarly, a region R_- is obtained by following the reverse flow from a neighborhood of $\mathbf{s}_-$. When there is a unique connecting orbit between $\mathbf{s}_+$ and $\mathbf{s}_-$, $R = R_+ \bigcap R_-$ is a region with the trivial Conley index. In the right portion, two valid regions R_1 and R_2 are obtained by using different sizes of the neighborhoods of $\mathbf{s}_-$. R_2 is preferred since it is larger.

region Q in the domain, its images under the forward and reverse flow are defined as:

$$\Omega(Q) = \varphi(Q, [0, \infty)) \quad \Omega^{-1}(Q) = \varphi(Q, (-\infty, 0]). \tag{75}$$

To find an isolating block R containing $\mathbf{s}_+$ and $\mathbf{s}_-$ on which the cancelation can be performed, I begin with isolating neighborhoods M and N of $\mathbf{s}_+$ and $\mathbf{s}_-$, respectively. In general, $R = \Omega(M) \bigcap \Omega^{-1}(N)$ is an isolating neighborhood. If there exists a unique separatrix going from $\mathbf{s}_+$ to $\mathbf{s}_-$, then the Conley index of R is trivial [62] (Figure 36, left).

In practice, M and N are sets of triangles that enclose $\mathbf{s}_+$ and $\mathbf{s}_-$, respectively. $R_+ = \Omega(M)$ is obtained by performing region growing from M and following the flow forward. Similarly, $R_- = \Omega^{-1}(N)$ is obtained by performing regions growing from N and following the flow backward. We need the following definition:

Definition 9.3.2. *Given a vector field V and a polygonal region R, a boundary edge e is an* **exit** *for the* **forward** *flow with respect to V if $\max_{p \in e}(N_p \cdot V_p) > 0$. Similarly, e is an* **exit** *for the* **backward** *flow with respect to V if $\min_{p \in e}(N_p \cdot V_p) < 0$. N_p is the outward normal to the region at point p.*

If V is a piecewise linear vector field on a boundary edge e, then e is an exit edge for the *forward* flow with respect to V if $\max(V(v_1) \cdot N_e, \quad V(v_2) \cdot N_e) > 0$. Also, e is an exit

edge for the *backward* flow with respect to V if $\min(V(v_1) \cdot N_e, \quad V(v_2) \cdot N_e) < 0$. Here, N_e is the outward normal to the region along edge e.

Let M be the triangle that contains $\mathbf{s}_+$. Starting from M, I construct $\Omega(M)$ by adding one triangle at a time and keeping track of the behavior of the flow on the boundary edges of $\Omega(M)$. A new triangle can be added only by crossing an exit edge. The region growing process continues until there are no more exit edges, i.e., the flow enters $\Omega(M)$ everywhere on its boundary.

$\Omega^{-1}(N)$ is constructed in a similar fashion by starting from N and following the flow backward. However, the choice of N is a delicate issue and its choice affects the shape of $\Omega^{-1}(N)$ and subsequently R. Due to the limited resolution of the underlying mesh, R needs to be as large as possible, so long as its Conley index remains trivial. A linear search is performed on the length of the outgoing separatrices of $\mathbf{s}_-$ such that the covering triangles form N. The right image in Figure 36 illustrates the effect of following these separatrices to varying lengths.

To replace the flow inside R, I use the flow smoothing operation described in Section 9.2 on R. As described before, this operation tends to simplify the vector field topology. The numerical results seem to indicate that flow smoothing is efficient for singularity pair cancelation as long as the region R has a reasonable shape.

Flow rotation is crucial for the success of singularity pair cancelation. If the original vector field has high curl around $\mathbf{s}_+$ as in the case of a divergence-free flow, the connecting orbit between the singularities may not even exist. Figure 37 demonstrates how a center and saddle are canceled (upper-left). The flow is first rotated by 90° into a curl-free vector field (upper-right). The center becomes a source and there is now a connecting orbit between the source and the saddle. Next, the source and the saddle are canceled. Finally, a compensating rotation of −90° is performed (lower-left).

9.3.2 Singularity Movement

Moving a singularity $\mathbf{s}$ to a new location allows the user to control the position of the singularities in a vector field. To my knowledge, this is the first time such an operation has

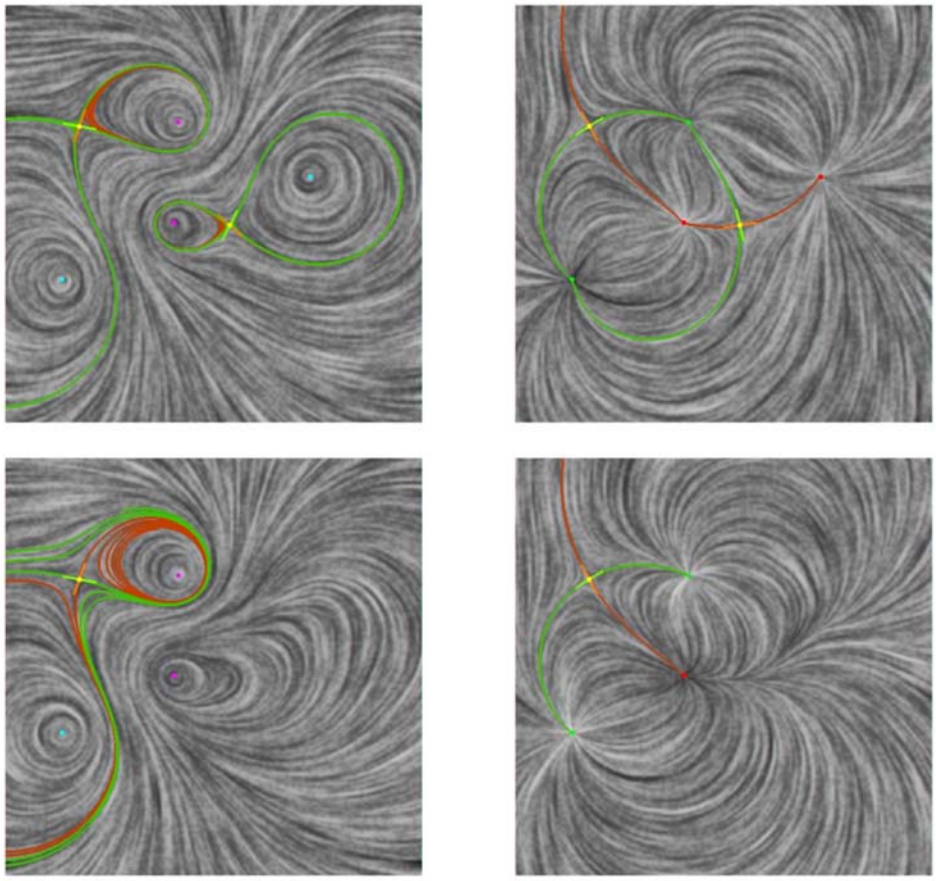

Figure 37: This figure shows how flow rotations help overcome the numerical difficulties associated with high curl in a vector field. A singularity pair cancelation is performed on a vector field (upper-left) to obtain a new vector field (lower-left). The vector field is first rotated by 90° (upper-right), followed by a source/saddle pair cancelation (lower-right) before a compensating rotation is performed (lower-left).

been proposed and implemented. Similar to singularity pair cancelation, this operation is designed to affect only the intended singularity. Through flow reflection and flow rotation, the problem of moving a singularity is reduced to moving a source, which is again carried out in two stages based on ideas from Conley index theory.

Consider the problem of moving a source. First, a region R is constructed such that it encloses the connecting orbit for the current location $\mathbf{s_{old}}$ and the desired new location $\mathbf{s_{new}}$ under the current vector field V. By construction, R has the Conley index of a source, and it does not contain any other singularities either in its interior or on its boundary (case (b) in Figure 27). Second, the vector field inside R is modified such that the new flow has only one singularity at $\mathbf{s_{new}}$ (Figure 38).

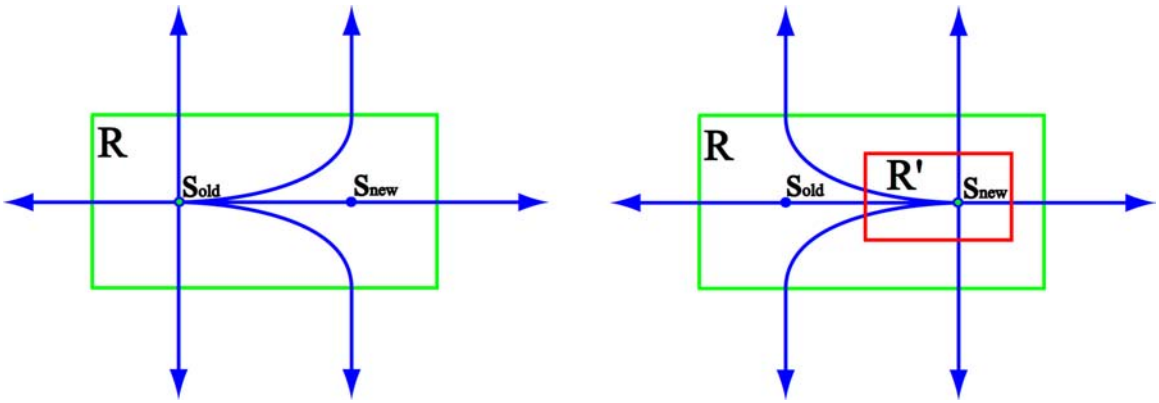

Figure 38: This figure illustrates the concept of moving a source from $\mathbf{s_{old}}$ to $\mathbf{s_{new}}$. A region R is found to enclose both $\mathbf{s_{old}}$ and $\mathbf{s_{new}}$ such that R has the Conley index of a source. Then a small region R' is found to enclose $\mathbf{s_{new}}$ and vector values are assigned to the boundary of R' such that it forces a source at $\mathbf{s_{new}}$. For region $R - R'$, flow smoothing operation produces a new vector field without any singularity.

Let $R = \Omega(M) \bigcap \Omega^{-1}(N)$. As before, M is a small neighborhood of $\mathbf{s_{old}}$ and N is a neighborhood of $\mathbf{s_{new}}$. To ensure $\mathbf{s_{new}}$ is in the interior of R, another point $\mathbf{s}'$ is located such that it is on the forward trajectory from $\mathbf{s_{new}}$ under V. Let us consider the trajectory J of $\mathbf{s}'$ under the flow $R_{\frac{\pi}{2}}(V)$. J serves the same purpose as the outgoing separatrices of the saddle in pair cancelation. Let N be the largest segment on J that makes R an isolating block with the Conley index of a source. This ensures that R is a wide region.

Let T be the triangle that contains $\mathbf{s_{new}}$. Vector values are assigned at the three vertices of T to force a source at $\mathbf{s_{new}}$. The region $R' = R \setminus \{T\}$ has two boundaries. The flow enters R' from the inner boundary and leaves at the outer boundary. R' therefore has the trivial Conley index (see Figure 27(e)), and flow smoothing inside R' usually produces a vector field without singularities.

Moving a center or a saddle is considerably more difficult than moving a source. First, finding a connecting orbit between the saddle and a regular point is numerical unstable. Moreover, finding a connecting orbit between a center and a regular point is almost impossible. To solve these numerical issues, I make use of flow rotations and reflections to make singularity movement applicable to any first-order singularity. If $\mathbf{s_{old}}$ is a saddle, I use a flow reflection to turn it into a source. If the vector field has high curl around $\mathbf{s_{old}}$, then I rotate the vector field so that the flow is converted to a vector field that has little curl

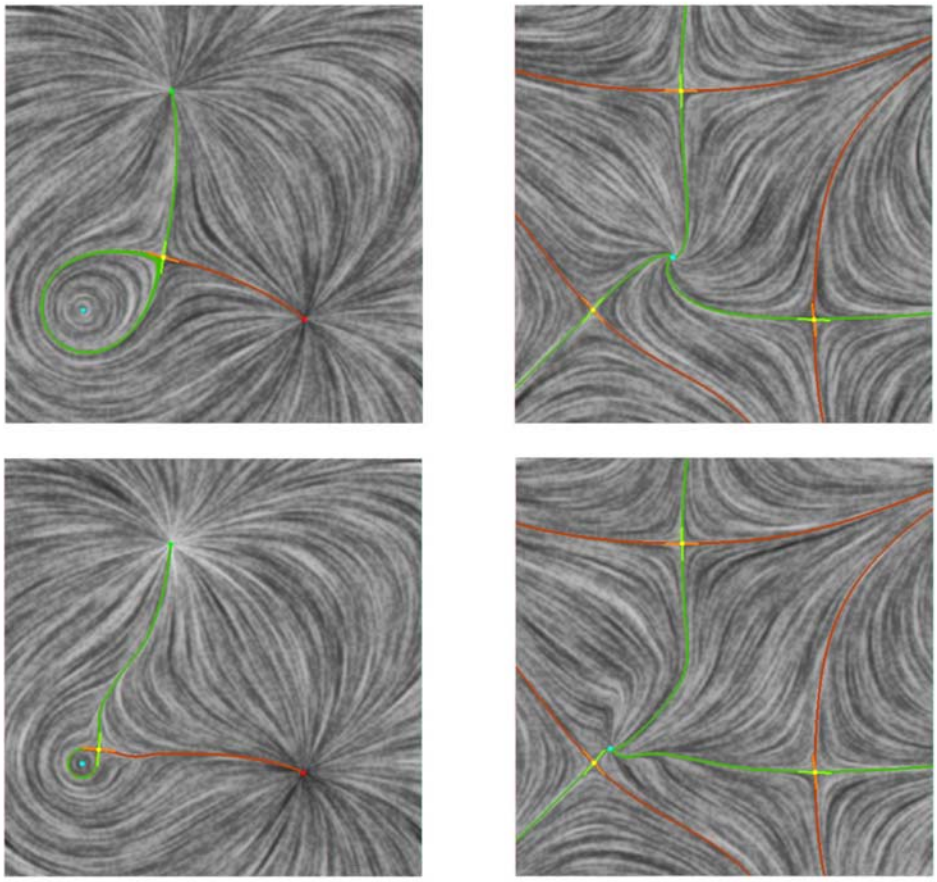

Figure 39: This figure shows how flow reflections help overcome the numerical difficulties associated with saddles. A singularity movement operation is applied to a vector field (upper-left) to obtain a new vector field (lower-left). The vector field is first reflected so that the saddle becomes a source (upper-right), followed by a source movement (lower-right) before a compensating reflection is performed (lower-left).

at $\mathbf{s_{old}}$. This simplifies the process of locating the connecting orbit between $\mathbf{s_{old}}$ and $\mathbf{s_{new}}$. Figure 39 provides an example of moving a saddle in a vector field (upper-left). First, a flow reflection is applied (upper-right) and the saddle is turned into a source. Next, the source is moved (lower-right) before a compensating reflection is applied (lower-left).

CHAPTER X

VECTOR FIELD DESIGN FOR SURFACES

Now that I have presented my design techniques for planar domains, I will describe how to extend these methods to mesh surfaces. As in the case of planar domains, the system for surfaces consists of three stages: initialization, analysis, and editing. However, there are several difficulties that must be overcome. First, tangent planes of a mesh surface are discontinuous at the vertices and edges. Therefore, the definition of vector field continuity from smooth manifolds does not apply to meshes. Second, a general curved surface lacks a global parameterization. Yet, we need a surface parameterization to compare tangent vectors that are defined at different locations.

In Section 10.1, I describe a definition for *vector field consistency* for mesh surfaces. The basic idea is to use the continuity of trajectories to define the consistency of a vector field. Next, I borrow ideas of the *exponential map* and *parallel transport* from differential geometry to set up correlations between tangent vectors defined at different parts of the surface. The correlations are used for two purposes. First, I extend the construction of basis vector fields from planar domains (Section 8.1) to surfaces by parallel transporting tangent vectors from the location of the design element to anywhere on the surface. Second, I adapt the piecewise linear approximation from planar domains (Section 8.2) to mesh surfaces by parallel transporting vector values from any vertex to its 1-ring neighborhood. The piecewise approximation results in a continuous surface vector field, and it supports efficient vector field analysis and editing operations on mesh surfaces.

10.1 Vector Field Consistency

For planar domains, the concept of vector field continuity is well-defined because any two vectors can be compared regardless of their locations. This is no longer true for a general surface since the tangent planes at different locations are distinct and there is not an obvious

and consistent way to correlate them without a global parameterization. Furthermore, the tangent planes of mesh surfaces are often discontinuous across the vertices and the edges. In order to construct continuous vector fields, I first propose a definition of vector field continuity for mesh surfaces.

Let us revisit Definition 7.0.2. For a continuous vector field, a point is a singularity if it has a zero vector value. Otherwise, it is regular. For surfaces, zero vectors at different locations can be identified regardless of locations, which allows us to define singularities for vector fields defined on surfaces. On the other hand, the fundamental theorem of Ordinary Differential Equations gives us a picture of what happens near a regular point [4]:

Theorem 10.1.1. *Let V be a continuous vector field defined in $D \subset \mathbb{R}^n$. If $\mathbf{p}_0 \in D$ is a regular point of V, then there exists a neighborhood U of $\mathbf{p}_0$ and a homeomorphism $h : U \to \mathbb{R}^n$ which carries each piece of a trajectory lying on U onto a straight line of $\mathbb{R}^n$ parallel to the X-axis.*

In other words, near a regular point $\mathbf{p}_0$, it is possible to warp the space such that the nearby trajectories are parallel and space-filling. I use this property as the definition for *vector field consistency* (continuity) for a regular point on a mesh.

Definition 10.1.2. *Let V be a vector field defined on a mesh surface $\mathbf{M}$. V is* **consistent** *at a point $\mathbf{p}_0 \in \mathbf{M}$ if one of the following situations is true:*

(a) For any path $\gamma : [0, 1) \to \mathbf{M}$ such that $\lim_{t\to 1} \gamma(t) = \mathbf{p}_0$, $V(\gamma(t))$ is well defined and $\lim_{t\to 1} V(\gamma(t))$ exists, then $\lim_{t\to 1} V(\gamma(t)) = 0$. In this case, $\mathbf{p}_0$ is a **singularity**.

(b) There exists a neighborhood U of $\mathbf{p}_0$ and a homeomorphism $h : U \to \mathbb{R}^2$ which carries each piece of a trajectory lying in U onto a straight line in $\mathbb{R}^2$ parallel to the x-axis. In this case, $\mathbf{p}_0$ is **regular**.

In other words, a consistent vector field on a mesh surface should exhibit the same local behaviors as those defined in a plane. Notice that as with curves on a continuous surface, it makes sense to discuss the continuity of the trajectories of a vector field. Figure 40 illustrates a consistent vector field V defined over the 1-ring neighborhood of a vertex O with a positive Gaussian curvature. The vector values are zero at A, B, C, and D. $V(O)$

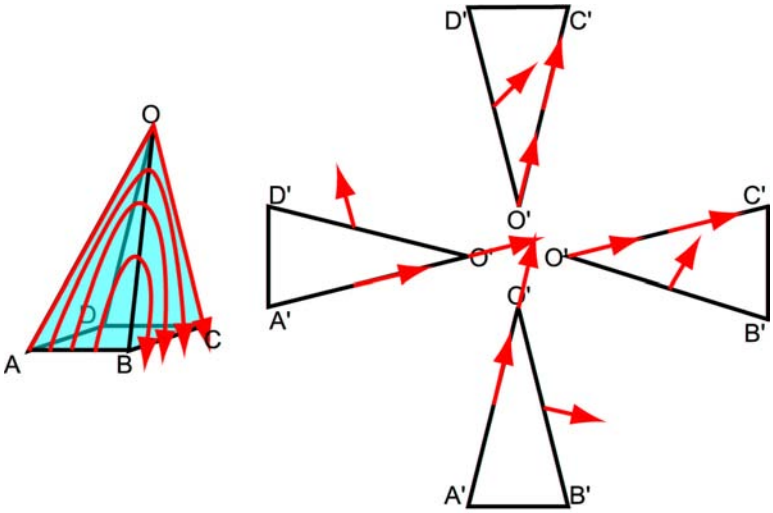

Figure 40: This figure illustrates a consistent vector field V defined in the 1-ring neighborhood of a vertex O with a positive Gaussian curvature. The vector values at A, B, C, and D are zero, and $V(O)$ is in the direction of $\overrightarrow{OC}$. Through the piecewise interpolation scheme (Section 10.3.1), the vector values are obtained for the edges and the interiors of the triangles (right, only directions are illustrated). Notice that V induces a family of non-intersecting and space-filling trajectories in the 1-ring neighborhood of O (left).

is in the direction of $\overrightarrow{OC}$. With the piecewise interpolating scheme that I am going to describe in Section 10.3.1, I produce a consistent vector field (right, only direction are shown). Notice that V induces a family of non-intersecting and space-filling trajectories in the 1-ring neighborhood of O (left).

In the remaining sections of this chapter, I will describe a vector field design system for mesh surfaces. Similar to planar domains, the system employs a three-stage pipeline: creating an initial vector field, analysis, and editing.

10.2 Basis Vector Fields for Initialization

For planar domains, the user creates an initial vector field by specifying singular and regular elements. The elements are converted into basis vector fields and summed. To extend this to surfaces, I present an approach in which surface basis vector fields are directly constructed from design elements. This approach is based on the ideas of *geodesic polar maps* and *parallel transport*.

Recall that in the planar case, a design element O is converted into a global basis vector

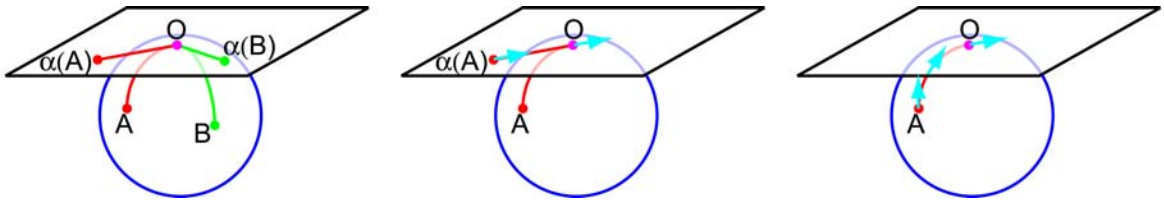

Figure 41: My three-step algorithm (from left to right) for creating a surface basis vector field from a user-specified constraint O. First, the surface is parameterized using a *geodesic polar map* with respect to O. This parameterization is denoted as α. Second, the basis vector field is computed inside the tangent plane at O with the polar coordinates from α. Finally, the vectors are *parallel transported* along shortest geodesics on the surface.

field through Equations 43 or 46. This is extended to surfaces through the following three-step process that is illustrated in Figure 41. First, I compute a *geodesic polar map* with respect to the location of O (left). This step assigns every point A on the surface with a pair of coordinates (x_A, y_A). This can seen as building a global parameterization for the surface using the tangent plane at O. Next, (x_A, y_A) are substituted into Equation 43 or 46 to obtain a tangent vector value W_A defined at O (middle). Finally, W_A is *parallel transported* along the shortest geodesic connecting O to A to obtain a tangent vector at A (right). The three-step process is based on several ideas from classical differential geometry, namely, *geodesics*, *exponential maps*, and *parallel transport*. I will review each of these in turn.

A geodesic on a curved surface is a locally shortest and straightest curve. It is a generalization of a straight line in the plane. Starting from a point $\mathbf{p}$ on the surface, there is a geodesic in every tangent direction $\overrightarrow{v}$. Denote this geodesic by $\gamma_{\mathbf{p},\overrightarrow{v}}$. A point $\mathbf{q}$ on $\gamma_{\mathbf{p},\overrightarrow{v}}$ with a distance ρ from $\mathbf{p}$ can be identified by the coordinates (ρ, θ). Here θ is the angular coordinate of $\overrightarrow{v}$ with respect some local frame at $\mathbf{p}$. In the plane, the coordinates reduce to the familiar polar coordinates. While it is termed an *exponential map* in differential geometry, several graphics papers have refer to this as a *geodesic polar map* [98, 68]. I adopt the latter name in this thesis. On a curved surface, a geodesic polar map is neither bijective nor continuous. For example, on the Earth, a geodesic polar map with respect to the North Pole will have discontinuity at the South Pole. However, since the focus of a design element is in a nearby region, the geodesic polar map with respect to the design element meets our needs.

In differential geometry, parallel transport is used to correlate tangent vectors defined at different locations using a geodesic that connects them. Formally,

Definition 10.2.1. *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be two points on a smooth manifold* S*, and let* $\gamma : [0, 1] \rightarrow S$ *be a geodesic such that* $\gamma(0) = \mathbf{p}$ *and* $\gamma(1) = \mathbf{q}$*. Furthermore, let* $V_{\mathbf{p}}$ *and* $V_{\mathbf{q}}$ *be tangent vectors defined at* $\mathbf{p}$ *and* $\mathbf{q}$*, respectively. Then* $V_{\mathbf{p}}$ *and* $V_{\mathbf{q}}$ *are said to be* **parallel** *with respect to* γ *if the oriented angle between* $\gamma'(0)$ *and* $V_{\mathbf{p}}$ *equals that between* $\gamma'(1)$ *and* $V_{\mathbf{q}}$*. Furthermore,* $V_{\mathbf{q}}$ *is said to be the* **parallel transport** *of* $V_{\mathbf{p}}$ *along* γ*.*

In the above definition, γ gives rise to an orthonormal linear map between $TM_{\mathbf{p}}$ and $TM_{\mathbf{q}}$, the tangent planes at $\mathbf{p}$ and $\mathbf{q}$. This map is a *transport* function and is denoted by $f_{\mathbf{pq}}$.

For a design element d, let $\alpha_d : S \rightarrow \mathbb{R}^2$ be a geodesic polar map with respect to d, and let $f_{d\mathbf{p}} : TM_d \rightarrow TM_{\mathbf{p}}$ be the transport function along a geodesic $\gamma_{d\mathbf{p}}$. Then the surface basis vector field $W(\mathbf{p})$ corresponding to d is constructed as:

$$W(\mathbf{p}) = f_{d\mathbf{p}} V(\alpha_d(\mathbf{p})) \tag{76}$$

In the above equation, V is evaluated according to Equations 43 or 46. For the purpose of building the geodesic polar map α_d and computing the transport function $f_{d\mathbf{p}}$, we need to compute a geodesic from any vertex on the surface to the design element d. In the following section, I will describe how to create a consistent vector field everywhere on the surface by interpolating the vector values defined at the vertices.

Assume that the design element d is situated inside a triangle T. I first compute the geodesic distance function g_d with respect to d for every vertex using the fast marching method [47]. The value of g_d at a vertex is the ρ component of the geodesic polar map. To construct the θ component in the ideal situation, one needs to perform particle tracing from a vertex in the opposite direction of $\bigtriangledown g_d$, the gradient vector field of g_d. However, performing particle tracing for every vertex is expensive. In addition, $-\bigtriangledown g_d$ often has sinks other than d. To overcome these problems, I make use of a two-region approach in which the angular component θ is computed directly *only* within a surface disk surrounding d such

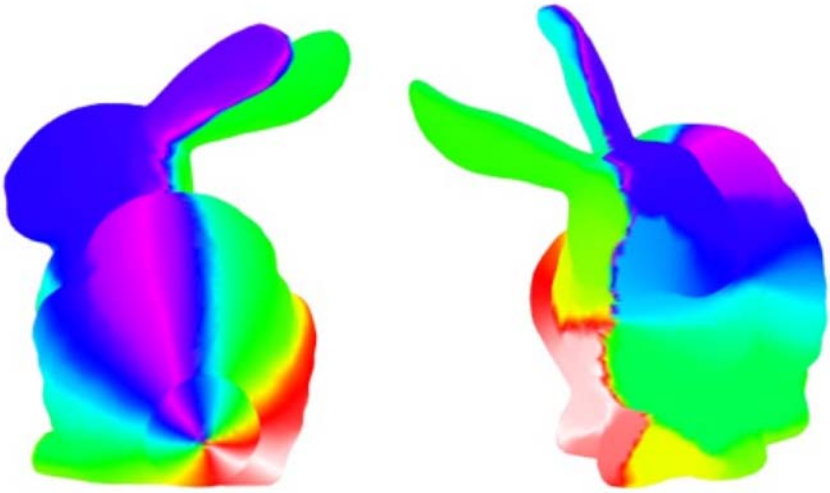

Figure 42: This figure shows the polar map that corresponds to the seed point on the bunny's tail. Notice the angular coordinate in the polar map makes sense near the seed. Also, there are seams in this map.

that the disk contains a user-specified percentage of the total vertices in the mesh. I use 25% for this application.

For a point $\mathbf{p}$ inside the disk, it is projected onto the tangent plane at d to obtain θ. For a vertex $\mathbf{p}$ outside the disk, I perform particle tracing from $\mathbf{p}$ in the direction of $-\bigtriangledown g_d$ until it hits any edge $e = \mathbf{rs}$. If $\theta(\mathbf{r})$ and $\theta(\mathbf{s})$ are both known, then $\theta(\mathbf{p})$ is obtained by linearly interpolating between $\theta(\mathbf{r})$ and $\theta(\mathbf{s})$. If particle tracing from $\mathbf{p}$ fails to reach any edge, then there is not a shortest geodesic between $\mathbf{p}$ and d. In this case, a random θ value is assigned to $\mathbf{p}$. Although this may seem to have created discontinuities in the vector field, let us recall that the vector values are only computed at the vertices at this stage. In the next section, I will describe a piecewise interpolating scheme in which a continuous vector field is created based on the values defined at the vertices. Figure 42 illustrates this map with the center at the bunny's tail. The colors are used to encode the angular coordinates. Notice the map is regular near the seed, but the seams often cause additional singularities at other positions.

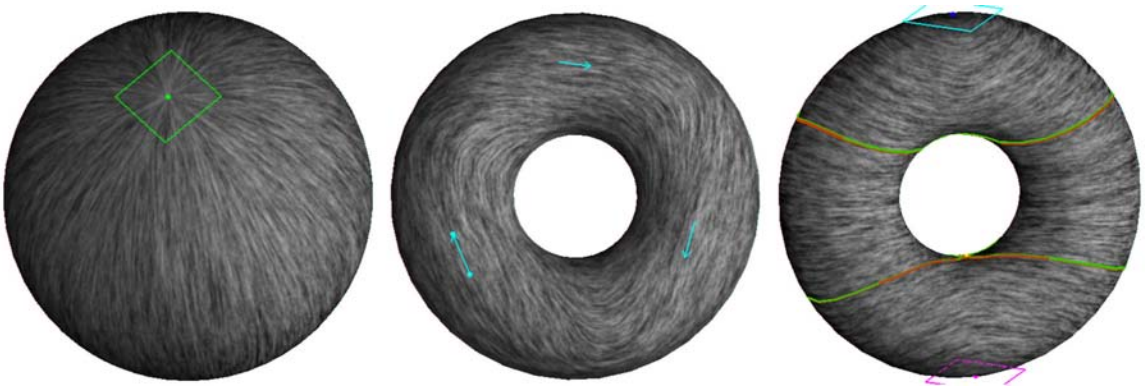

Figure 43: Design elements for creating an initial vector field. From left to right: a dipole vector field on a sphere using a source element, a singularity-free vector field on a torus with three regular elements, and another vector field on the torus with a clockwise center element and a counter-clockwise element. Notice the surface basis vector fields are very efficient for creating initial vector fields.

Given a geodesic polar map, a tangent vector can be parallel transported to a vertex along a geodesic. This completes the construction of a basis vector field. Figure 43 provides three example vector fields created in this manner: a dipole vector field on a sphere with a single source element (left), a singularity-free vector field on a torus with three regular elements (middle), and another vector field on a torus with a clockwise center element and a counter-clockwise center element (right).

Let me stress that this is not the only way to create basis vector fields. In van Wijk's visualization tool [95], an element is translated into a 3D vector field before being projected onto the surface. Constrained optimization [93, 97] is another way of producing an initial vector field with desired behaviors. Praun et al. [70] propose a vector field propagation approach in which a vector value is defined inside one face of the mesh surface. Through region growing, the vector value for a new triangle is obtained by computing the average tangent vectors at its neighboring triangles that are already part of region. This vector is then projected onto the face.

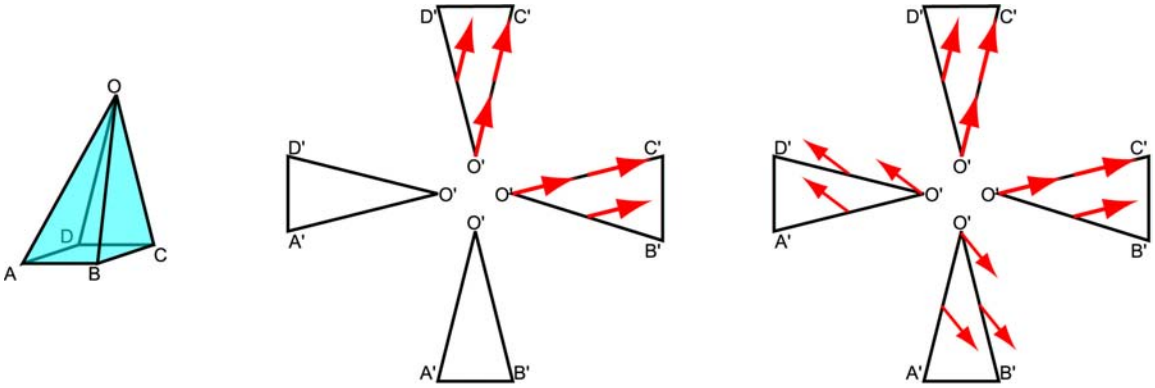

Figure 44: This figure illustrates that the piecewise linear representation does not produce continuous vector fields on mesh surfaces. The vector values are zero at A, B, C, and D. The vector value at O is in the direction of $\overrightarrow{OC}$. The piecewise linear representation and vector field consistency along edge OD and OB eventually lead to vector field discontinuity along edge OA.

10.3 Piecewise Approximation and Analysis for Vector Fields on Meshes

Once the vector values are obtained at the vertices of a mesh, I use a piecewise interpolating scheme to construct a continuous vector field on the mesh surface. Unfortunately, the piecewise linear approximation used for planar vector fields does not produce consistent vector fields on mesh surfaces. Figure 44 illustrates the problem for a vertex O and its 1-ring neighborhood (left). The vector values are zero at A, B, C, and D. The vector value $V(O)$ is in the direction of $\overrightarrow{OC}$. With the piecewise linear representation, the vector values at the midpoints of edge OD and OB are fixed (middle). Due to the continuity constraints across edges, the vector values at the middle points OA in triangle $\triangle ODA$ and $\triangle OAB$ lead to inconsistencies (right). The problem is due to the angle deficit caused by the discontinuity of tangent planes at the vertices and across the edges. However, we need consistent vector fields for control over vector field topology.

Stam [87] overcomes this problem by using subdivision surfaces, whose tangent planes are continuous everywhere. However, for most geometric processing operations, subdivision surfaces incur higher computational costs than polygonal meshes. In this thesis, I construct a consistent vector field directly on mesh surfaces with a new piecewise interpolating scheme

(see Figure 40 for an illustration). This scheme is a generalization of the piecewise linear representation from the planar case, and it allows fast and efficient analysis and editing of vector fields on meshes.

10.3.1 Piecewise Approximation

As in planar domains, a vector field V on a mesh surface is represented by assigning vector values $\{W_1, W_2, ..., W_n\}$ at the mesh vertices $\{v_1, v_2, ..., v_n\}$. However, we cannot simply perform linear interpolation since the W_i's are in general not co-planar. Furthermore, without a surface parameterization, tangent vectors that are defined at different vertices are not correlated. To overcome these problems, I first define a local parameterization for the 1-ring neighborhood of a vertex v_i. This parameterization allows the parallel transport of W_i to any point $\mathbf{p}$ inside v_i's 1-ring neighborhood. Let μ_i be such transport function (which I will soon describe). Then, for a point $\mathbf{p}$ inside a triangle $T = \{v_{T_1}, v_{T_2}, v_{T_3}\}$ whose barycentric coordinates are $(\alpha_1, \alpha_2, \alpha_3)$, Equation 47 can now be rewritten as the weighted sum of the tangent vectors that are parallel transported from the three vertices:

$$V(\mathbf{p}) = \sum_{j=1}^{3} \alpha_j \mu_{T_j}(W_{T_j}, \mathbf{p}) \tag{77}$$

This results in a consistent vector field over the mesh surface based on the values of W_i. Before I describe the parameterization and the transport function in detail, we need the following definitions [68].

Definition 10.3.1. *Let M be a polyhedral mesh representing a closed curved surface. Let v be a vertex with incident triangles T_i $(i = 1, ..n)$. Let θ_i be interior angle of T_i at v. Then the* **total vertex angle** *$\theta(v)$ at v is given by:*

$$\theta(v) = \sum_{i=1}^{n} \theta_i \tag{78}$$

$2\pi - \theta(v)$ is the **Gaussian curvature** *at v. A vertex v is called*

1. **spherical** *if the Gaussian curvature is > 0.*
2. **Euclidean** *if the Gaussian curvature is $= 0$.*

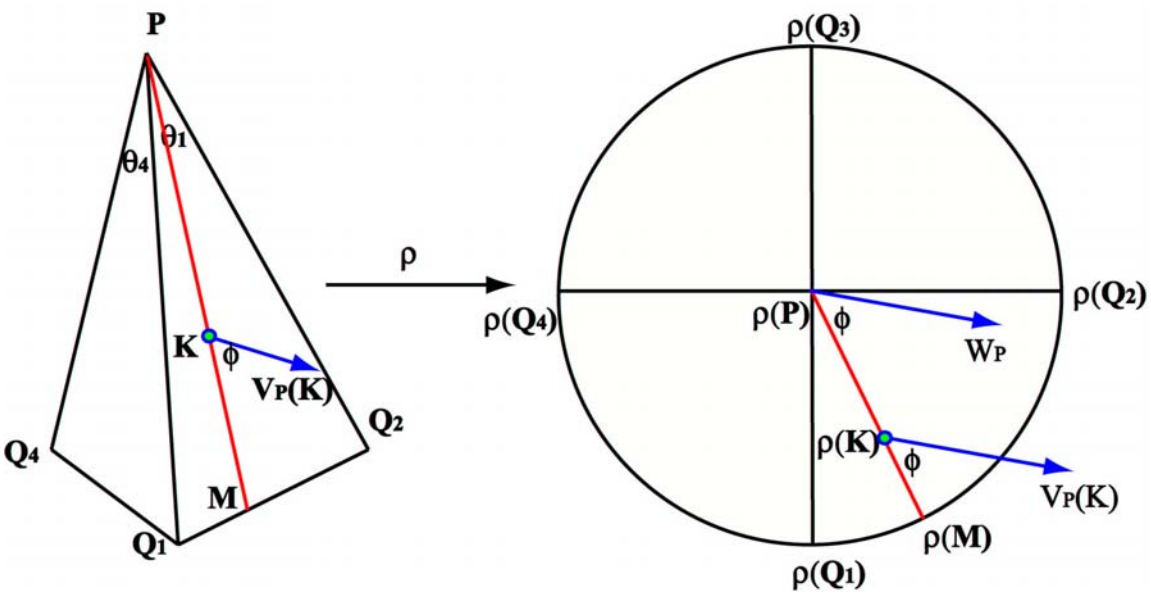

Figure 45: This figure illustrates the idea of parallel transporting a vector W_i from a vertex $v_i = \mathbf{P}$ to a point $\mathbf{K}$ inside $\mathbf{P}$'s 1-ring neighborhood, R. First, I build a local parameterization ρ for R. Then, $W_{\mathbf{P}}$ is parallel transported to $\mathbf{K}$ along the ray $\overrightarrow{\rho(\mathbf{P})\rho(\mathbf{K})}$. This construction guarantees vector field consistency on mesh surfaces.

3. **hyperbolic** *if the Gaussian curvature is* < 0.

Spherical and hyperbolic vertices are **non-Euclidean**.

In Figure 45 (left), $\mathbf{P} = v_i$ is a vertex with the tangent plane $TM_{\mathbf{P}}$ (right). Its 1-ring neighborhood R consists of triangles $\triangle\mathbf{P}\mathbf{Q}_1\mathbf{Q}_2$, $\triangle\mathbf{P}\mathbf{Q}_2\mathbf{Q}_3$, ..., and $\triangle\mathbf{P}\mathbf{Q}_n\mathbf{Q}_1$ ($n = 4$). Let $\theta_i = \angle\mathbf{Q}_i\mathbf{P}\mathbf{Q}_{i+1}$, $\theta = \sum_{i=1}^{n}\theta_i$ and $r = \frac{\theta}{2\pi}$. Notice that $r = 1$ for vertices with a zero Gaussian curvature. In the right portion of this figure, let D be the unit disc in $TM_{\mathbf{P}}$ and let ρ be the following homeomorphism from R to D.

1. ρ induces a bijective mapping between the boundary of R and the boundary of the unit circle. For any point $\mathbf{M} \in \partial R$, ρ is a linear map from $\overrightarrow{\mathbf{PM}}$ to $\overrightarrow{\rho(\mathbf{P})\rho(\mathbf{M})}$.

2. ρ is a linear scaling on the angles between rays. If two rays emanating from $\mathbf{P}$ have an oriented angle of θ, then the angle between their images in $TM_{\mathbf{P}}$ is $r\theta$.

Note that this construction is similar to the "geodesic polar map" used by Welch and Witkin [98] for free-form shape design, with a minor difference: in their setting the parameterization domain is a polygon, not the unit disc as in our case.

To transfer W_i to a point $\mathbf{K}$ inside triangle $\mathbf{Q}_i\mathbf{P}\mathbf{Q}_{i+1}$, I first locate the ray $\overrightarrow{\mathbf{PM}}$ that contains $\mathbf{K}$. Let ϕ be the counter-clockwise angle between W_i and the ray $\overrightarrow{\rho(\mathbf{P})\rho(\mathbf{M})}$, then I define $\mu_i(W_i, \mathbf{p})$ as the vector at $\mathbf{K}$ such that the angle between $\mu_i(W_i, \mathbf{p})$ and $\overrightarrow{\mathbf{PM}}$ equals ϕ. Furthermore, $|\mu_i(W_i, \mathbf{p})| = |W_i|$.

As a parameterization, ρ does not distinguish between points inside a triangle or on an edge. Consequently, vector field continuity is automatically guaranteed there. Furthermore, the continuity of ρ ensures the continuity of the resulting vector field at the vertices. For planar domains, r is equal to one everywhere and this approximation becomes the piecewise linear representation for planar vector fields (Section 8.2).

The piecewise approximation scheme induces a vector field W that is a continuous and non-linear inside each triangle T minus three arbitrarily small corners, each around a vertex. Therefore, W can be seen as being defined over a hexagon that is arbitrarily close to T. Along each edge of the triangle, W is linear in terms of length. Along the sides where the corners are cut, W is linear in terms of vertex angle. Locating singularities of W is rather difficult under this setting. Furthermore, the Poincaré index for a hexagon can be ± 2, which implies possibly two first-order singularities or one second-order singularity inside T. This makes topological control more difficult. I perform a four-fold triangle subdivision on the input mesh. Basically, the mid-point of every edge in the original mesh becomes a new Euclidean vertex since it total vertex angle is 2π. This means that every triangle in the subdivided mesh can have at most one non-Euclidean vertex, and analysis becomes tractable. From now on, I will assume the input mesh already satisfies this requirement.

10.3.2 Analysis

In this section, I describe how to perform analysis using the piecewise approximation from the previous section. Let $T = \triangle QRS$ be a triangle with exactly one non-Euclidean vertex Q. Then the vector field V as constructed in Equation 77 is linear on RS and along any ray emanating from Q. Furthermore, W is a continuous vector field defined on T minus an arbitrarily small corner near Q, i.e., a quadrilateral as illustrated in Figure 46. Let $V(R)$ and $V(S)$ be the vector values of V. At Q, we need a direction to determine the vector

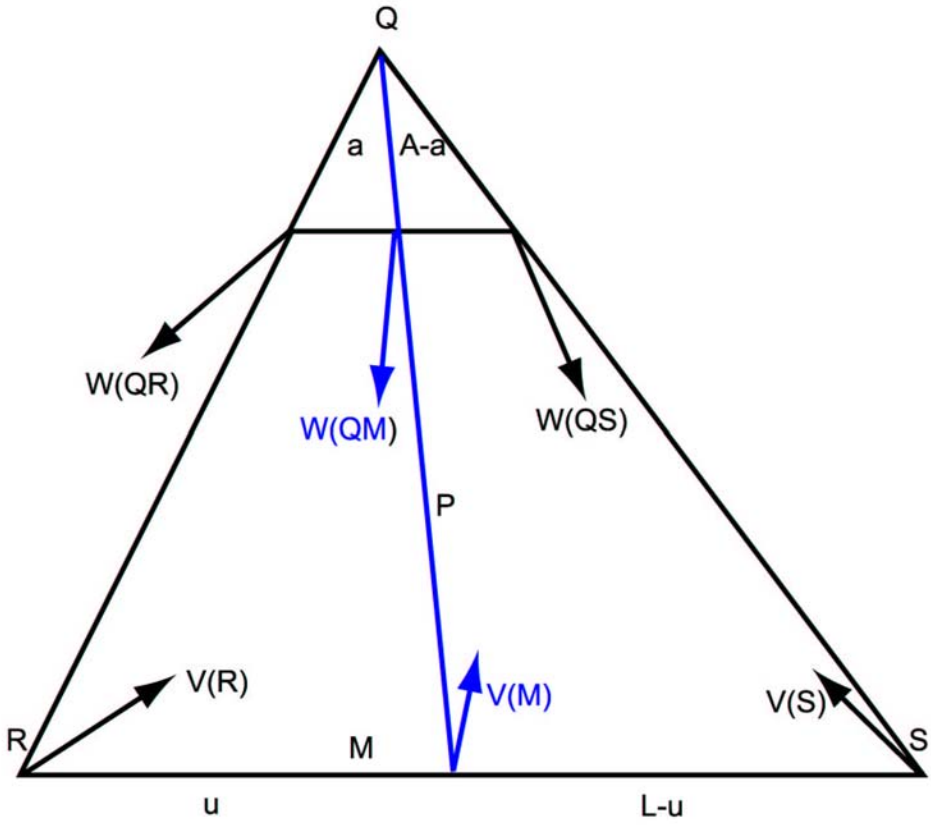

Figure 46: This figure illustrates how to determine the location of a singularity inside a triangle under the piecewise interpolating scheme. Here, Q is the only non-Euclidean vertex inside the triangle.

value. Let W denote the vector field of V along an arbitrarily small line segment near Q such that $W(QR)$ and $W(QS)$ are vector values in the direction $\overrightarrow{QR}$ and $\overrightarrow{QS}$.

10.3.2.1 Local Linearization

The Jacobian is used to compute the curl and divergence, to determine the type of singularities, and to find the incoming and outgoing directions for a saddle. Under the piecewise linear approximation for planar domains, the Jacobian is constant inside each triangle. For mesh surfaces, I compute the Jacobian for every point M_0 inside a triangle T as follows. First, two points M_1 and M_2 are selected inside T such that they are sufficiently close to M_0, and $\overrightarrow{M_0M_1}$ and $\overrightarrow{M_0M_2}$ are not co-linear. Next, I construct a linear vector field V' such that $V'(M_i) = V(M_i)$ for $i = 0, 1, 2$. Finally, the Jacobian of V at M_0 is approximated by the Jacobian of V'.

10.3.2.2 The Curl and Divergence

The curl and divergence at a point P could be approximated using local linearization. Instead of this approach, the curl and divergence for a triangle can be accurately computed as follows:

$$curl(T) = \frac{(W_{QR} + V_R) \cdot N_{\overrightarrow{QR}}}{2} |\overrightarrow{QR}| + \frac{(V_R + V_S) \cdot N_{\overrightarrow{RS}}}{2} |\overrightarrow{RS}| + \frac{(V_S + W_{SQ}) \cdot N_{\overrightarrow{SQ}}}{2} |\overrightarrow{SQ}| \quad (79)$$

$$div(T) = \frac{(W_{QR} + V_R) \cdot D_{\overrightarrow{QR}}}{2} |\overrightarrow{QR}| + \frac{(V_R + V_S) \cdot D_{\overrightarrow{RS}}}{2} |\overrightarrow{RS}| + \frac{(V_S + W_{SQ}) \cdot D_{\overrightarrow{SQ}}}{2} |\overrightarrow{SQ}| \quad (80)$$

Here, N_e and D_e are the outward normal and forward tangent vector of an edge e, respectively.

Remark 10.3.2. *The total curl and divergence is zero for any closed 2-manifold. The piecewise approximation maintains this for mesh surfaces by construction. My numerical results also support this.*

10.3.2.3 The Poincaré Index

The Poincaré index of a triangle $T = \triangle QRS$ is computed for the quadrilateral (the triangle minus an arbitrarily small corner) as illustrated in Figure 46. Along each side, the vector field continuously and monotonically interpolates the vector values at the end points. Let us consider $W(QR)$ and $W(QS)$ as vector values defined at the ends of an arbitrarily small edge. The total index angle will be in $(-4\pi, 4\pi)$. Therefore, T has at most one first-order singularity.

Remark 10.3.3. *The piecewise approximation maintains the Poincaré-Hopf theorem, and my numerical results also support this.*

10.3.2.4 Singularities

If the Poincaré index of T is not zero, there must be a singularity inside T. To locate the singularity P, I perform a binary search for a point $M \in \overrightarrow{RS}$ such that $V(M)$ and

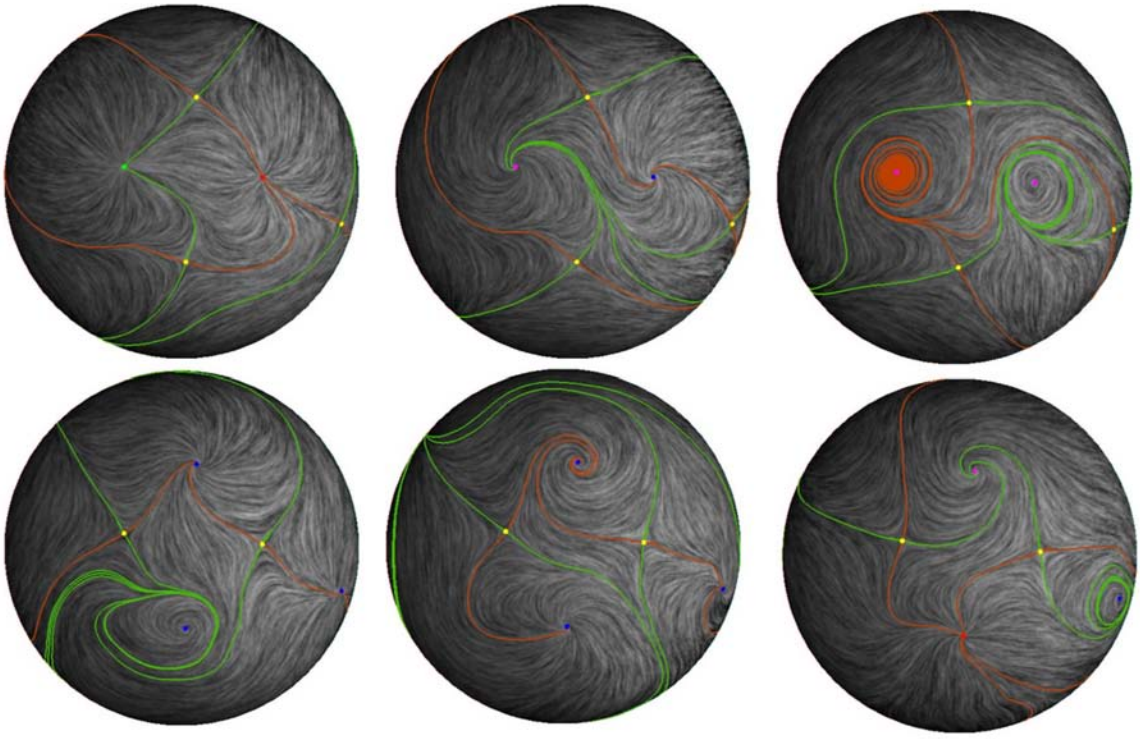

Figure 47: Flow rotations are applied to a vector field on a sphere. Top row, from left to right are rotations of 0°, 45°, 90°. Flow reflections are applied to these vector fields to produce corresponding images in the bottom row. Observe that flow rotations maintain the number, the location, and the Poincaré index of the singularities. On the other hand, flow reflection maintain the location of singularities while negating the sign of their Poincaré indices.

$W(QM)$ are co-linear and they point in opposite directions. Let $W(QM) = -\alpha V(M)$, then $P = (1-m)Q + mM$ $(m = \alpha/(\alpha+1))$ is the singularity. Local linearization at P is used to determine its type, and in the case of a saddle, its incoming and outgoing directions.

10.3.2.5 Separatrices

The Runge-Kutta method that I have used for computing separatrices for planar vector fields (Section 8.2) can be adapted to mesh surfaces. Polthier and Schmies have also suggested a fourth-order Runge-Kutta method for computing trajectories for a continuous vector field on mesh surfaces [68]. Figure 47 show some examples vector fields on a sphere along with their topological skeletons. The analysis is performed using the piecewise approximation that I have described in this section.

10.4 Editing Operations

While the main concepts for editing operations on a surface remain the same as those for planar domains, some changes need to be made to reflect the differences in vector field representation and the complexity of the surface geometry such as curvature and higher genus.

10.4.1 Matrix Actions on Flows

To perform flow rotations on a mesh surface, I simply rotate the vector values of W_i's in the tangent planes at each vertex. Since the transport functions are orthonormal transformations between the tangent planes (Section 10.3.1), rotating W_i's by an angle of θ results in a rotation of the same angle inside every triangle and edge. Therefore, flow rotations maintain the number, the location, and the Poincaré index of the singularities. Furthermore, for any point inside a triangle, Equations 67, 68, and 69 remain valid.

Notice that flow rotations for surface vector fields do not require any global parameterization. On the other hand, flow reflections require that the local frames and reflection axes at every vertex be correlated. I make use of a global polar map to parallel transport this information. Flow reflections swap the sign of the Poincaré indices. In addition, they may create some additional singularities due to the singularities in the field of local frames and the field of reflection axes. Regardless of these issues, the resulting vector field still satisfies the Poincaré-Hopf theorem due to the piecewise interpolating scheme that I describe in Section 10.3.1. In Figure 48, a flow reflection on a dipole vector field on the sphere converts the source and the sink into a pair of saddles. In addition, some singularities are created at places where the field of local frames and the field of reflection axes are degenerate. The total index of the resulting vector field still equals the Euler characteristic of the sphere. Performing flow reflection twice with the same field of reflection axes results in the original vector field.

Figure 47 illustrates the effects of performing flow rotations and flow reflections on a vector field. In the top row and from left to right are flow rotations of 0°, 45°, and 90°. In the bottom row, flow reflections are applied to the corresponding vector field in the

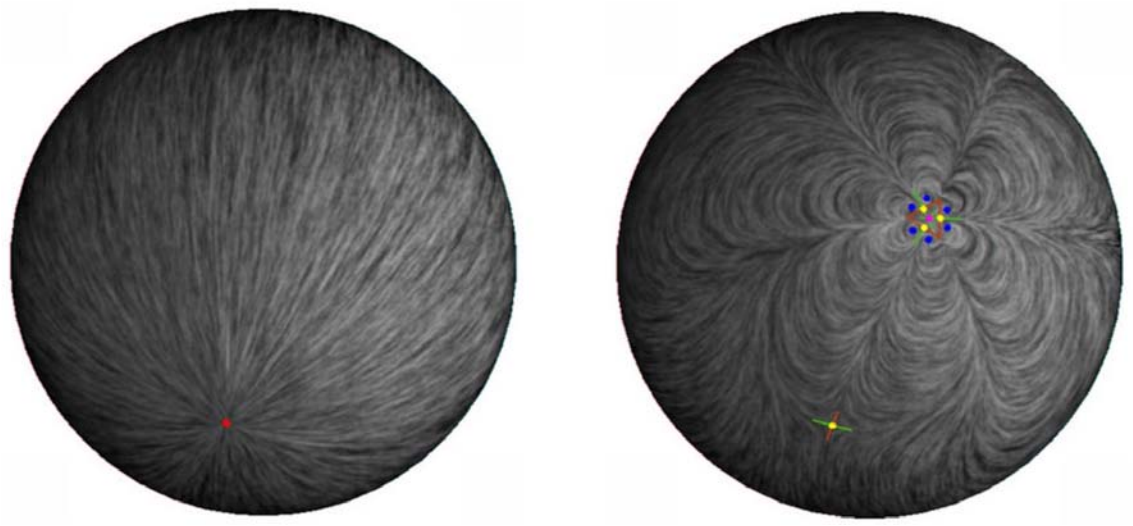

Figure 48: Flow reflection on surfaces will often create additional singularities due to the discontinuity in the field of local frames and the field of reflection axes. In this figure, the original vector field (left) has exactly two singularities: a source and a sink. After a flow reflection operation is applied (right), the singularities become saddles. In addition, the flow reflection operation also creates a cluster of singularities near the regions of discontinuity for the reflection axes. However, the total Poincaré indices of the new vector field is still equal to the Euler-characteristic of the sphere, due to the piecewise interpolating scheme that I describe in Section 10.3.1.

same column in the top row. Notice that a flow rotation does not change the number, the location, and the Poincaré index of the singularities. A flow reflections does not change the location of the singularities, but it negates the sign of the Poincaré indices. Compare this figure with Figure 32.

Similar to the planar domains, I use flow rotations to overcome the numerical difficulties associated with regions of high curl, and I use flow reflections to properly handle saddles.

10.4.2 Flow Smoothing

Flow smoothing is used to smooth a vector field inside a user-specified region. It is also used for topological editing operations, such as singularity pair cancelation and singularity movement.

Similar to planar domains, the vector values are held fixed at the boundary vertices of the user-specified region R. The vector values for the interior vertices are solved through a vector-valued Laplacian equation. I have implemented two variations of flow smoothing for

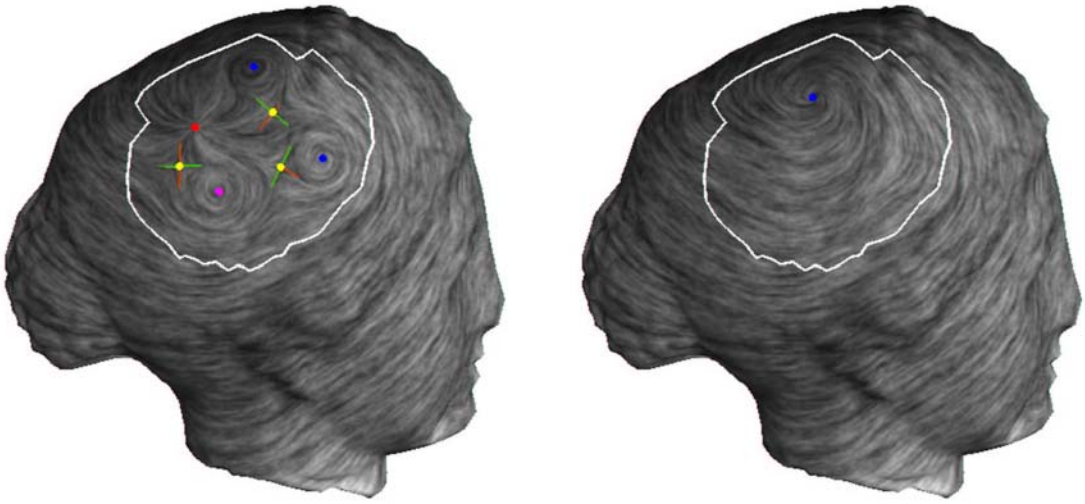

Figure 49: A vector field defined on the Venus model is smoothed inside a user-specified region (the smaller region bounded by the white loop). Notice that the new vector field inside this region has a simpler topology.

meshes. In the first variation, the vector field is obtained by smoothing a 3D vector field and projecting it onto the tangent planes at each vertex. The second approach parameterizes R based on some planar domains and perform smoothing in this domain. I will describe the second method in more detail.

Let T be a center triangle of R, i.e., it is a local maxima of the distance function from the boundary of R. Let $\mathbf{p}_0$ be the center of T. I first construct the geodesic polar map α with respect to $\mathbf{p}_0$. Next, the vectors at each vertex of R are parallel transported to $\mathbf{p}_0$. The smoothing operation is then conducted in the tangent plane at $\mathbf{p}_0$ with respect to α, and the resulting vector field is parallel transported back to the vertices.

Both smoothing techniques provide similar results. However, theoretically speaking, the second approach seems more natural for vector-valued smoothing on mesh surfaces. Figure 49 illustrates the effects of flow smoothing using the second approach on a vector field defined on the Venus model.

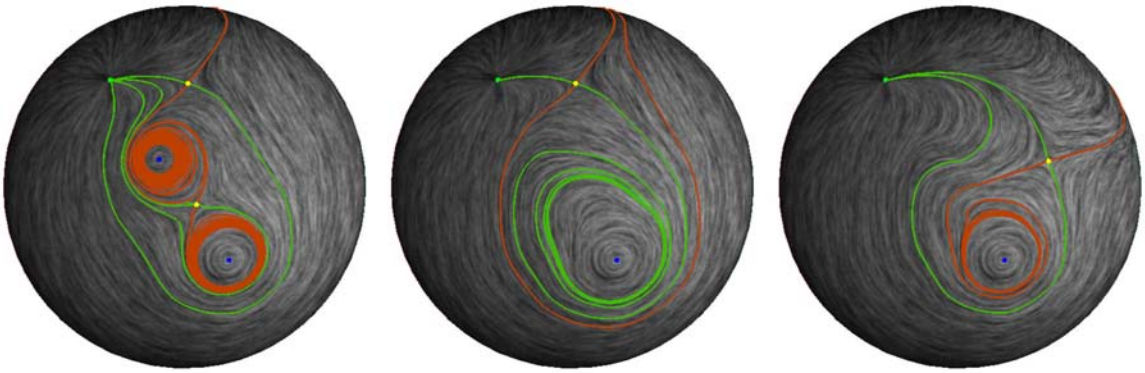

Figure 50: In this figure, a sequence of topological editing operations are performed on a vector field defined a sphere (left). First, a saddle and a center are canceled (middle). Next, the remaining saddle is moved (right).

10.4.3 Topological Editing Operations

Both singularity pair cancelation and singularity movement operations may require the following:

1. **Flow rotations and reflections** are used to overcome numerical instabilities associated with regions of high curl, and with saddles.

2. **Build isolating blocks** by performing region growing with respect to the flow. This requires us to determine whether an edge is an exit (Section 9.3.1). Recall that an edge e is an *exit* for a region R with respect to the forward flow if the flow leaves R from at least part of e. Similarly we have defined an exit edge with respect to the backward flow.

3. **Flow smoothing** is used to replace the flow in the interior of the isolating blocks.

While the concepts of topological editing operations for mesh surfaces are the same as those for planar domains, several changes need to be made. I have addressed the changes that are needed for the first and the last item in the list. The only remaining issue is how to determine for a triangle, which edges are forward exit edges and which are backward exit edges. Since the piecewise approximation guarantees that the vector field along an edge is always linear, it suffices to compute the dot products between the outward normal to the edge and the vector values at the vertices of the edge. Let T be a triangle and $e = QR$ be

an edge of T. Assume Q is a non-Euclidean vertex. Then as illustrated in Figure 46, the vector values at Q and R are W_{QR} and V_R, respectively.

In Figure 50, a series of topological editing operations are performed on a vector field defined on a sphere (left). Notice that these operations only affect the intended singularities. Also, a flow rotation is used to cancel a pair of center and saddle, and a flow reflection is used to move the remaining saddle.

CHAPTER XI

SOME GRAPHICS APPLICATIONS FOR VECTOR FIELD DESIGN ON SURFACES

All the vector fields shown in this thesis are created with my system. In addition, I have applied my vector field design system to several graphics applications: painterly rendering of images, pencil sketch illustration of smooth surfaces, and example-based texture synthesis. In this section, I will show the results.

11.1 Painterly Rendering

Painterly rendering refers to creating digital images that have the appearance of being painted. There are numerous published approaches to painterly rendering, and to review them all is beyond the scope of the thesis. These techniques have focused on providing the user with control over certain aspects of brush strokes (textures, styles) while automatically determining other aspects (base colors and orientations). In particular, image-based gradient fields have often been used to guide the orientation of brush strokes. While this may be appropriate for some parts of the image (near the feature lines), it often produces brush strokes with noisy orientations in areas with nearly uniform colors. Furthermore, it creates unnecessary constraints on the way that artists may express themselves. My goal for this application is to let the user control the brush stroke orientation through vector field design.

I use a level-of-detail approach by Hertzmann [38]. In this approach, a painting is created in a series of layers, starting with a rough sketch drawn with brush strokes of a large size. Then the sketch is painted over with brush strokes of gradually decreasing sizes at the places where signals of higher-frequencies are present. This approach is very fast and has high-quality. However, I make the following modification: instead of using the image gradient field to guide the brush stroke orientations, let the user create a synthetic vector

field with my vector field design system. To make this task fast and effective, I incorporate the painterly rendering algorithm into my vector field design system. In addition to viewing the vector field, the user can also switch to the painterly rendering that uses the current vector field. The results are interactively displayed as the user makes changes to the vector field. Figure 51 shows the painterly rendering results for two source images: a human's eye (top, Van Goth style) and a cat face (bottom, impressionism). For the human's eye, a center element was placed at the middle of the pupil, and a saddle element was placed at the corner of the eye. Two regular elements were placed along the eyebrow to ensure that the brush strokes do not cross the feature. With five elements, a vector field (top, lower-left) was produced that matches the main features in the image (the eye and the eyebrow). For the cat's face, two center elements with opposite orientations were placed at the middle of the eyes. A saddle element was placed underneath the nose and six regular elements were placed along the ears and the chin. The final high-quality painterly images in this figure were created off-line using the algorithm of Hays and Essa [36].

11.2 Non-Photorealistic Illustration of Surfaces

There has been much work in creating hatch-based illustrations of surfaces, and to review all of them is beyond the scope of this thesis. Girshick et al. [27] have shown that the principle curvature fields are good at conveying shapes. Traditional techniques often make use of principle curvature directions to guide the hatch field. Hertzmann and Zorin [39] present an efficient algorithm for approximating the principle curvature fields over the mesh by local surface fitting, which leads to high-quality pen-and-ink style of rendering of 3D shapes. Praun et al. [72] propose a new way of creating hatching for 3D surfaces. They treat the problem of hatch-based illustration as performing texture synthesis on surfaces. This leads to a real-time hatching system in which the user has the option to guide the orientation of hatches with a vector field on a 3D model.

Similar to the image gradient field for painterly rendering, the principle curvature fields are rather noisy for regions where the principle directions are not prominent. In this work, I allow the user to guide the hatch field by designing a vector field through my system.

I have modified van Wijk's image based flow visualization technique [95] to create a fast and efficient non-photorealistic illustration of surfaces. Figure 52 shows the results of applying this technique to the feline, Venus, the horse, and the dragon. The vector field used for the dragon model in the lower-right image was obtained by rotating the vector field from the lower-left image by 60°.

11.3 Example-Based Texture Synthesis

Example-based texture synthesis refers to creating patterns on surfaces based on a given input image of a texture. Praun et al. [70] propose "lapped textures" in which the surface is partitioned into overlapping regions and each region receives a portion of the input image. This method is fast, but causes seams due to surface partition. For textures that contain only high frequencies, the seams are relative unnoticeable. Another class of methods [93, 97] perform synthesis on surfaces directly without creating seams. For any point on the surface, its color is copied from the pixel in the input image that provides the best neighborhood match based on a distance criterion. For both types of synthesis methods, a vector field is used to provide local orientation and scale. For the surface synthesis methods [93, 97], the same vector field is also used to determine the synthesis order.

Figure 53 shows the results of applying my surface vector field design system to texture synthesis on a bunny and a feline model. The texture synthesis method is based on [93, 97]. The two vector fields used for the bunny are: a sink element at the tail and a source element on the forehead (upper-left), and a 60° rotation of a dipole (a source element and a sink element) on the visible side of bunny (upper-right). Notice that the spiralling in the second vector field near the singularities on the side of the bunny is evident in the texture in the upper-right image. Figure 3 shows the visualization of these vector fields (middle and right). The lower portion of Figure 53 shows the feline with a tiger stripe pattern that is guided by two different vector fields. Both vector fields produce reasonable results.

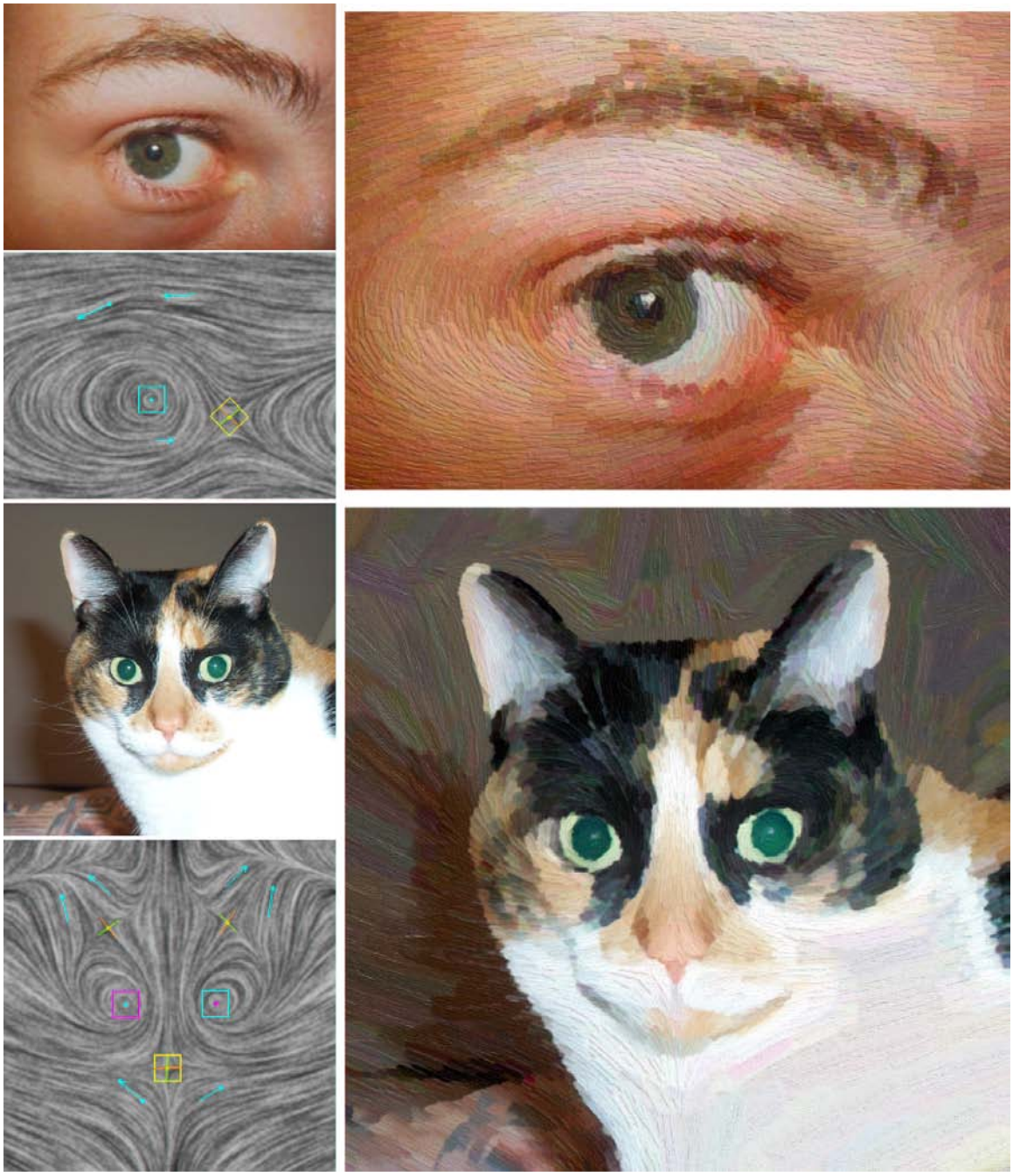

Figure 51: This figure shows the results of applying my planar vector field design system to painterly rendering. The upper and lower portions show the painterly rendering of two input images: an eye and a cat. The input vector fields are also shown. Note that the high-quality images shown here are produced off-line with the approach of Hays and Essa [36].

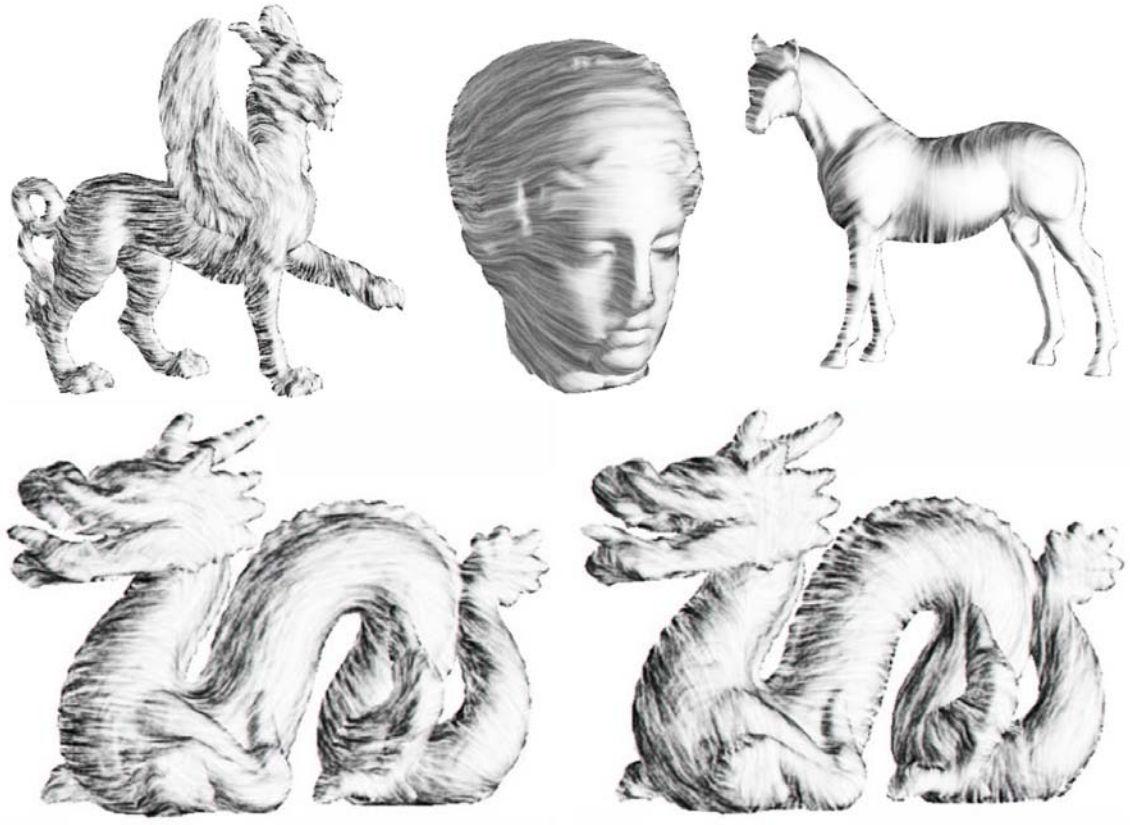

Figure 52: This figure shows the results of applying my vector field design system to non-photorealistic illustrations of surfaces. The pencil style rendering algorithm is based on van Wijk's image-based flow visualization technique [95]. The vector field used for the dragon image in the lower-right portion was obtained by rotating the vector field from the lower-left portion by 60°.

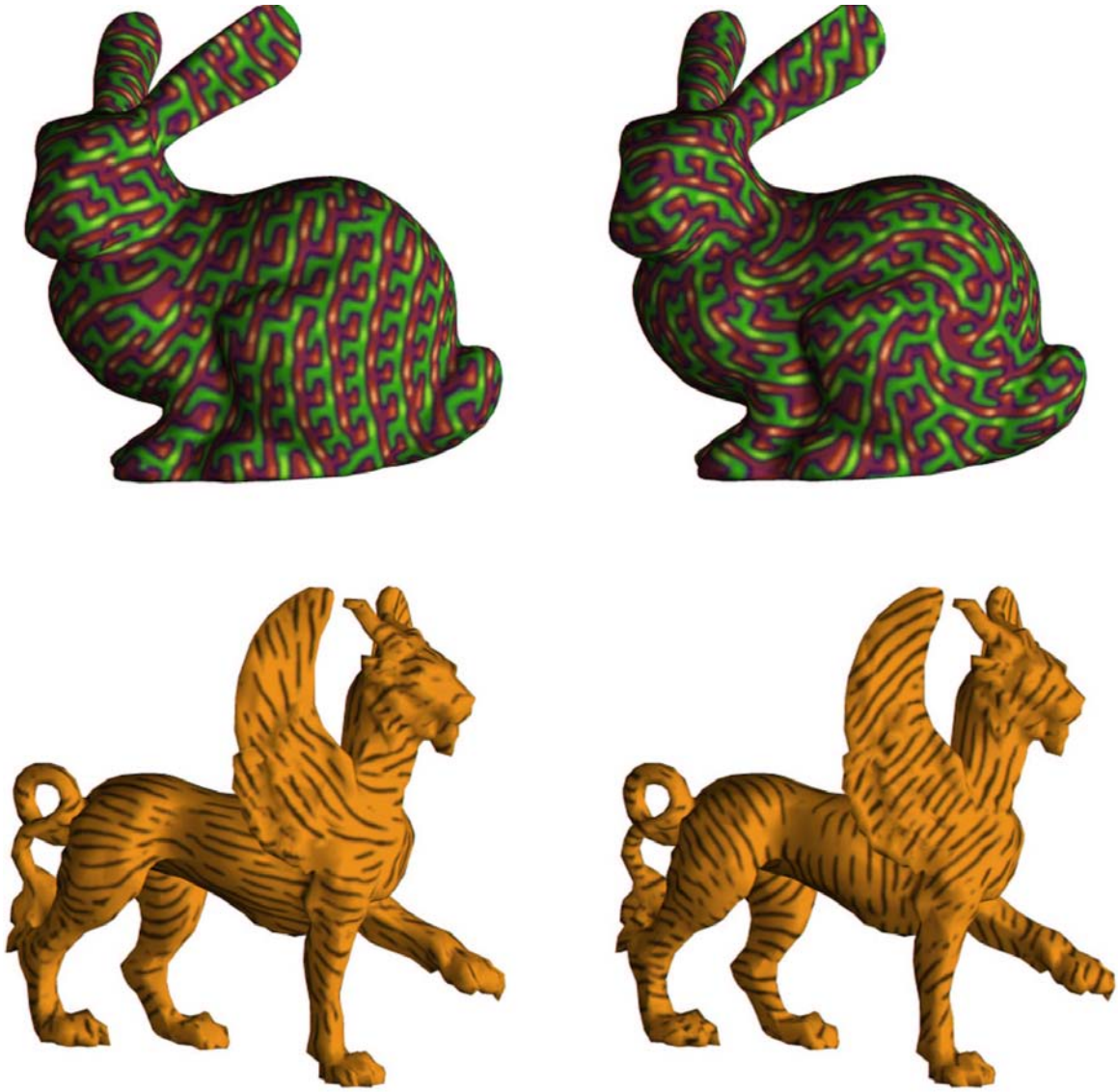

Figure 53: This figure shows the results of applying my surface vector field design system to texture synthesis. Two vector fields are used for each model: the bunny, and the feline. Notice that the singularities in the vector fields lead to the breakup of the synthesis patterns (upper-right). Also, the spirals around these singularities are obvious in the synthesis result. For anisotropic textures, different vector fields lead to different visual appearance (compare the bunny images and the feline images, respectively).

CHAPTER XII

CONCLUSION AND FUTURE WORK

In this chapter, I summarize my contributions in surface parameterization and vector field design and discuss some possible future research directions.

12.1 Surface Parameterization

12.1.1 My contributions

Surface parameterization is an important problem for geometric processing. High stretch and a large number of seams caused by parameterization will cause visual artifacts. In this work, I present an automatic surface parameterization method with the following contributions:

Patch creation: in which a surface is decomposed into a small number of "simple" shapes based on the features contained in the model. My patch creation method contains several new ideas:

1. *AGD*, the average geodesic distance function, is used to reveal the locations of the topological and geometric features contained in a surface. I also propose an efficient approximation of AGD on mesh surfaces.
2. *The embedded Reeb graph* is constructed by performing region growing over mesh surfaces and by keeping track of the topology of the boundaries. This graph can be used to break up handles and to segment large protrusions from the surface.
3. *Feature identification*, in which I have described an algorithm that locates a separating region for a given large protrusion. I have also presented a topological consistent technique that produces a short and smooth separating cycle based on the separating region.

4. *Genus reduction*, in which I construct locally short and smooth non-separating cycles to break up the handles.
5. *Baseball decomposition* is constructed for nearly spherical shapes in order to reduce stretch.

These ideas result in a small number of large patches, each of which can be unfolded with relatively little stretch.

Patch unfolding: I have described two enhancements over existing unfolding techniques:

1. The *Green-Lagrange tensor* is used to measure stretch and to guide the non-linear vertex optimization [78].
2. *Scaffold triangles* are used to allow the optimization of the boundary vertices without the need to check for global self-intersections.

The combination of these enhancements results in lower stretch during the unfolding stage.

Patch packing: I present a new packing technique that takes into account the natural orientations of the unfolded patches. This packing technique also keeps track of the usable space in a more "frugal" manner than previous techniques.

Image-based quality metric: I describe a quality metric for parameterization. This metric is based on the image differences between the original textured surfaces and the reconstructed ones from texture mapping. The metric implicitly takes into account a number of important issues for parameterization: stretch, seams, packing efficiency, smoothness, and surface visibility.

12.1.2 Interesting Future Directions

There are a number of interesting open research problems in this area.

12.1.2.1 Handling seams

Seams occur where the surface is cut, and they cause discontinuity in color variations when making a textured image. While the *push-pull* technique by Sander et al. [78] helps to

alleviate the problem, it does not eliminate the problem entirely. There have been few techniques that are targeted at reducing seams, and the work by Gu et al. [30] has the virtue of creating as few seams as possible. However, reducing seams may cause high stretch. Imagine a sphere that is cut along its shortest edge and unfolded. Sheffer and Hart [84] make use of visibility to reduce the artifacts caused by seams. It would be desirable to directly measure stretch and seams at the same time.

12.1.2.2 Quality Measure for Parameterization

The quality of a parameterization is application-dependent. For texture mapping, in which the size of the texture map is limited, stretch is of paramount importance. On the other hand, compression techniques require the tracking of seams. While many parameterization techniques exist and many measures have been developed for measuring stretch and seams, direct application-dependent quality measures have been lacking. The quality metric proposed in this thesis is a first attempt to address this issue. However, many questions remain. For instance, how are the existing stretch metrics, such as the Green-Lagrange tensor and the metric developed by Khodakovsky et al. [46], related to the image-based quality metric described in Section 5.2? Studying these quality measures may lead to a better understanding of the problem and to better parameterization techniques.

12.1.2.3 Better Understanding and Better Use of AGD

In this work, I make use of AGD to identify the centers of large protrusions. To segment protrusions, a different surface distance-based function is used. A question remains: do we fully understand AGD and its potentials as a shape descriptor? The answer is probably "No". For instance, how is AGD related to curvature? The answer to this question may lead to feature extracting methods that are directly based on AGD. I suspect that surface curvature information is embedded in the higher-order derivatives of AGD.

12.1.2.4 Other Global Feature Descriptors

Besides AGD, there are probably other surface-based functions that can be used to describe certain global characteristics of a surface. For instance, human facial features are small

protrusions if measured using AGD. Nonetheless, humans use these features to recognize each other. Furthermore, when some features are out of range, such as an extremely long nose or a pair of small eyes, people take notice very quickly. What are the measurements that humans use for these features?

12.1.2.5 Other Types of Parameterization

Recently, there have been renewed interests in non-planar parameterization. For instance, the work by Praun and Hoppe [71] and Gotsman et al. [28] deal with spherical parameterizations. For topological spheres, a spherical parameterization seems more appropriate than plane-based ones addressed in this thesis. However, there may still be the need for surface segmentation in order to reduce stretch. With the exception of a sphere, any closed orientable 2-manifold can be obtained by performing identification of $\mathbb{R}^2$ based on some groups. Using mathematical language, $\mathbb{R}^2$ is a covering space for these manifolds. I am interested in understanding the structures of such parameterizations and learning how the identification maps may help handle seams.

12.1.2.6 Parameterization as an Atlas

Surface parameterization is important for many applications, one of which is surface visualization. A complex surface often contains interiors and concavities that are difficult to see from outside viewpoints. I would like to investigate the use of parameterization for surface exploration purposes, giving the user the ability to navigate, orient and focus.

12.2 Vector Field Design

12.2.1 My Contributions

Vector field design on surfaces is an important problem that has received relatively little attention. In this work, I have made the following contributions to the area.

Applications and Requirements: I have identified a number of graphics applications, such as non-photorealistic rendering and texture synthesis, for which an input vector field is needed. Different vector fields can lead to different visual effects. I also propose a set of requirements for a vector field design system. Namely, the user can create a

large variety of vector fields (not some sub-class) with relatively little effort. Also, the user should be able to control the number and location of the singularities.

The System: I present a vector field design system for both planar domains and 3D mesh surfaces. To my knowledge, this is the first system that can produce continuous vector fields on a mesh surface and provide control over vector field topology. The system has a three-stage pipeline: creating an initial vector field, analysis, and editing. The editing operations are at the core of the system.

Technical Contributions: To make the system fully functional, I have introduced algorithms to resolve the following problems. Many of these problems are challenging by themselves.

1. *Vector field representation for mesh surfaces*: my piecewise approximation of vector fields for mesh surfaces is novel, and it enables us to perform analysis and editing operations (rotation and reflection, singularities pair cancelation and movement) for continuous vector fields on mesh surfaces. To my knowledge, existing pair cancelation techniques work only for planar domains.

2. *Basis vector fields on meshes*: I describe a new technique for efficiently producing surface basis vector fields that correspond to user specifications. The method is based on the concept of *geodesic polar maps* and *parallel transport*.

3. *Singularity movement*: I provide a new operation to control the location of the singularities in a vector field. Based on Conley index theory, I provide a unified framework for implementing both singularity pair cancelation and singularity movement. The region optimization technique that I describe helps to produce a smooth vector field after applying the editing operations.

4. *Matrix actions*: I use flow rotations to overcome numerical instabilities associated with sources and sinks of high curl, and I use flow reflections to handle saddles. I also extend these operations to surface vector fields.

12.2.2 Interesting Future Directions

There are several possible areas for future research.

12.2.2.1 Better Understanding of the Vector-Valued Smoothing Operations

In my implementation of the topological editing operations, I have used vector-valued smoothing operations. While such operations often simplify vector field topology inside a region, it is desirable to understand the conditions under which the smoothing operations are guaranteed to produce the desired results. For instance, under what conditions do such operations guarantee to produce a singularity-free vector field inside a region with the trivial Conley index? This question seems to be linked to the concept of *convex combination maps over a triangulation* [24], which have appeared in surface parameterization [22, 53], surface smoothing [88], solving Laplacian equations over mesh surfaces [91], and designing fair Morse functions over a surface [63]. Floater discusses the properties of convex combination maps extensively in [24]. The following result is relevant to the properties of the vector-valued flow smoothing operation.

Let the set of vertices of the triangulation T be $\{V_1, V_2, ..., V_n\}$, and let $\phi : T \rightarrow \mathbb{R}^2$ be a convex combination map over T. Then $\phi(T) \subset ch(\{\phi(V_1), \phi(V_2), ..., \phi(V_n)\})$, i.e., the image of T under ϕ is a contained in the convex hull formed by the images of the vertices of T.

For flow smoothing, the above statement translates to the following. Let R be a compact region, and assume vector values are defined for the boundary vertices of R. If the convex hull of vector values for boundary vertices in $\mathbb{R}^2$ does not include $(0, 0)$, then flow smoothing with non-negative weights will result in a singularity-free vector field inside R. Figure 54 provides an example in which flow smoothing does not produce a singularity-free vector field. It remains an open question that what is the necessary and sufficient condition for flow smoothing to produce a singularity-free vector field.

Besides the topological properties of vector-valued flow smoothing operations, it is also important to understand their impacts on the geometric properties of the "smoothed" vector fields. For instance, how does such an operation redistribute the curl and divergence of the vector field? Also, how do such operations affect a region near a center?

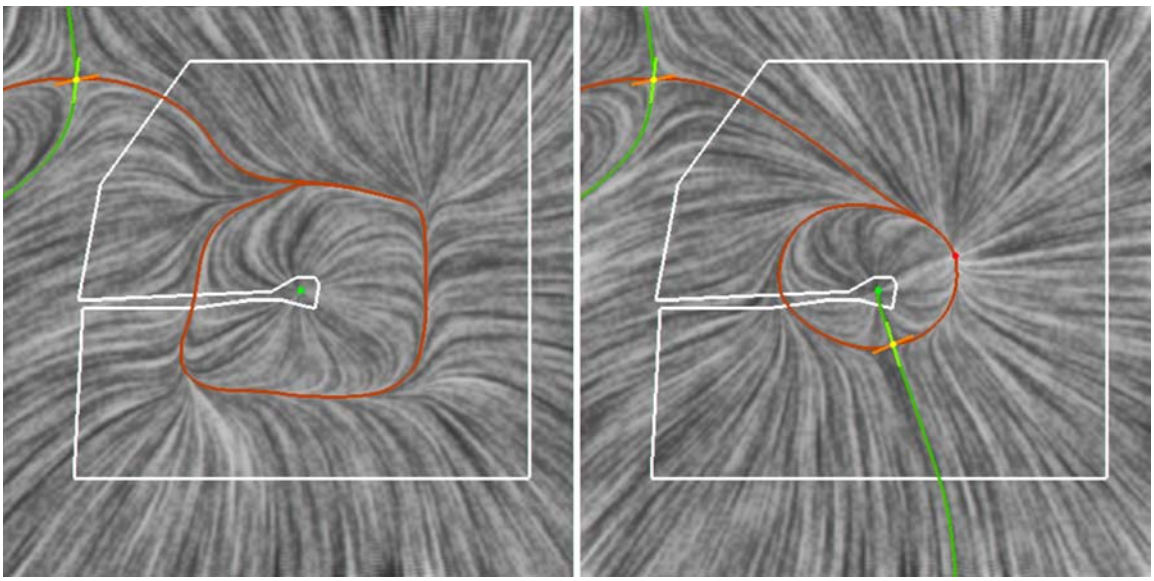

Figure 54: In this example, applying flow smoothing to the vector field (left, inside the white boundary) actually increases the number of singularities.

12.2.2.2 Automatic Singularity Pair Cancelations

In this work, I have let the user to select the singularity pair for cancelation. Sometimes it is desirable to have the system automatically select the singularity pair. For the gradient vector field of a Morse-Smale function, such techniques exist [20, 8]. The ideas behind these techniques are the "persistency" of the singularities. For a Morse-Smale function, the meaning of persistency is well-defined. However, it remains an open question how to measure the persistency of an arbitrary singularity or a periodic orbit in a vector field. Moreover, what is the persistency of a particular trajectory, whether singular or regular?

A related question is the following: given a 3D surface and given user-specified singularities, what is the "simplest" vector field that satisfies the constraints. If we assume the vector field has only linear singularities, then the Poincaré-Hopf theorem provides a lower bound.

12.2.2.3 Control over Separatrices and Limit Cycles

My work in this thesis has focused on controlling the singularities in a vector field. It is natural to ask for controls over the separatrices and limit cycles, which are also part of the vector field topology. Can we extend the concept of singular elements and allow the user

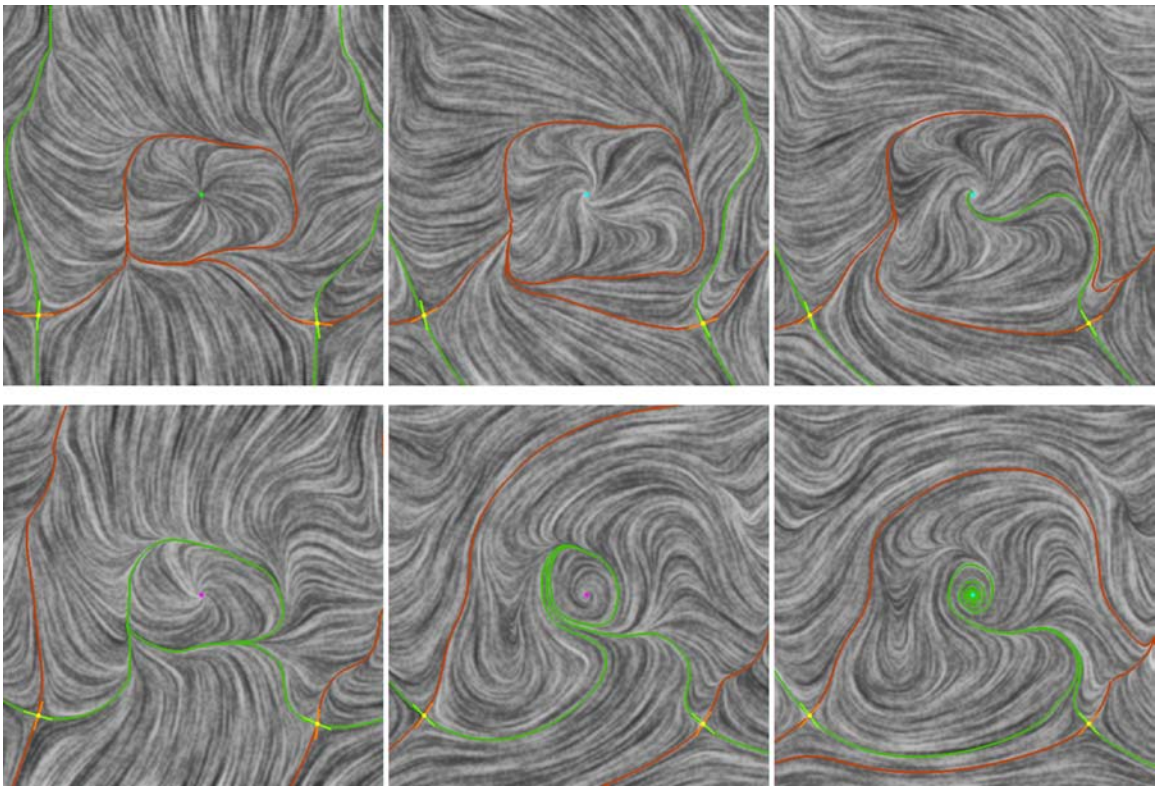

Figure 55: In this figure, a series of flow rotations are applied to a vector field (upper-left) with an attracting limit cycle. Going clockwise from upper-left, the rotations are 0° (upper-left), 45° (upper-middle), 60° (upper-right), 90° (lower-right), 105° (lower-middle), 150° (lower-left). Notice that the attracting limit cycle (upper-left) eventually becomes a repelling limit cycle (lower-left).

to create canonical separatrices and limit cycles? What editing operations are necessary to control them? Finally, and perhaps more importantly, what graphics applications will benefit from these operations?

As an example, in Figure 55, a sequence of flow rotations were applied to a vector field with an attracting limit cycle. Notice that the attracting limit cycle (upper-left) eventually became a repelling limit cycle (lower-left).

12.2.2.4 Other Editing Operations

Singularity pair cancelation can be seen as performing a particular type of bifurcation if one tracks the continuous change that is involved. Many other types of bifurcations exist. Figure 56 show two such examples: basin bifurcation (top) and homoclinic connection

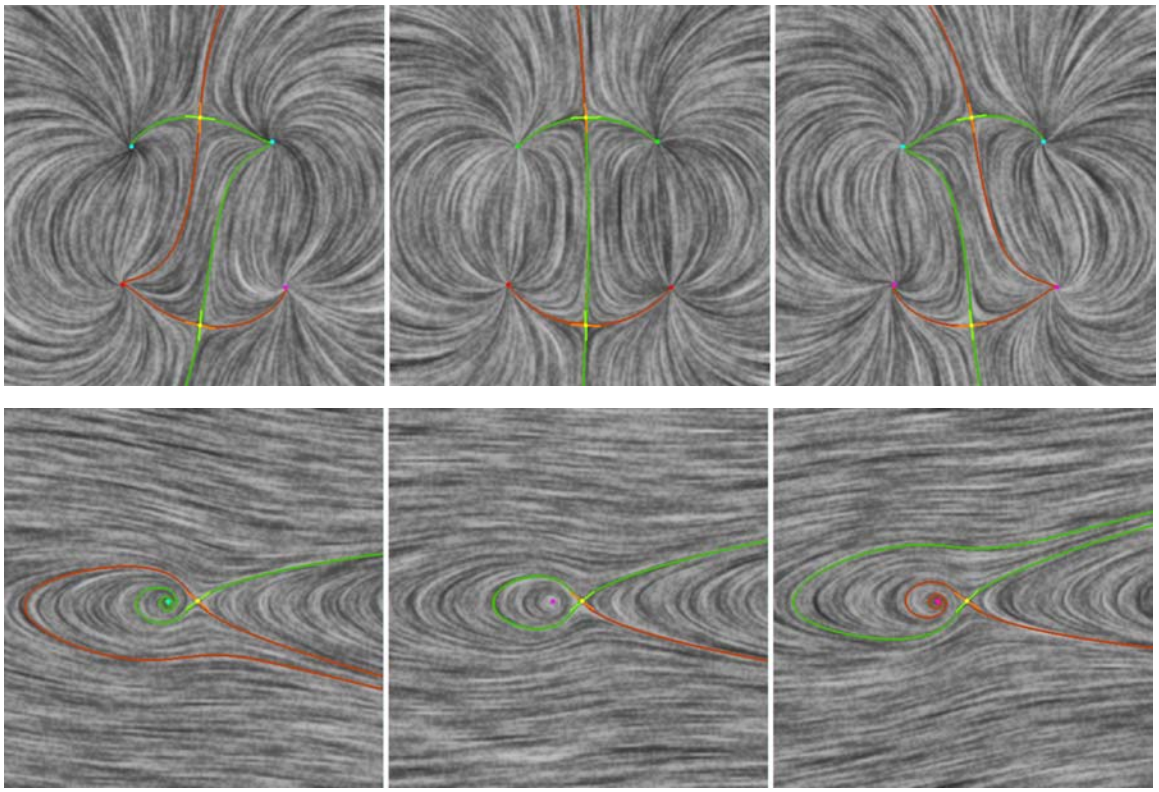

Figure 56: This figure illustrates two example bifurcations: basin bifurcation (top row) and homoclinic connection bifurcation (bottom row). In each example, one can continuously change the vector field shown in the left to the one shown in the right. Due to the different vector field topology, a bifurcation has to be happen during the continuous vector field change. The bifurcation point (shown in the middle) has a different vector field topology.

bifurcation (bottom). Bifurcations are interesting mathematically, and they also have applications for scientific visualization.

12.2.2.5 Vector Field Design with Techniques for Scalar Field Design

A surface vector field is the sum of a curl-free vector field and a divergence-free vector field (Helmholtz-Hodge decomposition [91]). Through a proper flow rotation, a divergence-free vector field can be turned into a curl-free vector field. Furthermore, under certain assumptions such as that the domain is simply connected, a curl-free vector field is the gradient of scalar field. Therefore, it seems possible that vector field design can borrow techniques from scalar field design [19, 63]. On the other hand, it is not obvious how the singularities in the vector field are related to the singularities in its curl-free (divergence-free) part.

12.2.2.6 Application-Dependent Initial Vector Fields

In this thesis, I have described a way in which a user designs a vector field to guide brush stroke orientations in painterly rendering, and to guide the hatch directions in non-photorealistic illustration of surfaces. While this approach gives the user much freedom, sometimes it makes sense to allow the user to start with an application-dependent vector field, such as the image-gradient field and principle curvature directions. I am exploring this possibility.

12.2.2.7 Vector Field Design for Other Domains and for Other Applications

Vector field design on surfaces might be extended to handle vector fields defined on volumes or other higher-dimensional data sets. Also, I am interested in identifying other applications for vector field design, such as fluid simulation and hairstyle design. Another important application is for educational purpose, in which students learn important concepts of vector fields through creating and manipulating vector fields and observing the changes.

APPENDIX A

PROPERTIES OF MATRIX ACTIONS ON FLOWS

In this appendix, let us consider the action on a planar vector field V by a 2×2 matrix M with a non-zero determinant. Let $A_M(V)$ be the resulting vector field.

Theorem A.0.1. *$A_M(V)$ has the same set of singularities as V.*

Proof. Since M has full rank, $Mv = 0 \iff v = 0$.

□

Theorem A.0.2. *$\forall$ two vectors v, w such that $|v, w| \neq 0$, we have $|Mv, Mw| \neq 0$.*

Proof. $|Mv, Mw| = |M||v, w| \neq 0$ since M has full rank and $|v, w| \neq 0$.

□

Theorem A.0.3. *Let $\mathcal{S}$ be the unit circle centered at the origin O. The action $\mu : \mathcal{S} \to \mathcal{S}$ by $\mu(P) := \frac{M\overrightarrow{OP}}{|M\overrightarrow{OP}|}$ is an* automorphism*, i.e., a self-homeomorphism for $\mathcal{S}$.*

Proof. Clearly μ is continuous. It suffices to show that μ is bijective, which follows directly from Theorem A.0.2.

□

Theorem A.0.4. *Let $\mathbf{p}_0$ be an isolated singularity of V with a Poincaré index η. Then $\mathbf{p}_0$ is also an isolated singularity of $A_M(V)$ with a Poincaré index $sgn(|M|)\eta$.*

Proof. Since $\mathbf{p}_0$ is an isolated singularity of V, there exists a disc Γ such that Γ is a neighborhood of $\mathbf{p}_0$ and V contains no singularities on $\Gamma - \mathbf{p}_0$. From Theorem A.0.1, we know that $\Gamma - \mathbf{p}_0$ does not contain any singularities of $A_M(V)$. Therefore, $\mathbf{p}_0$ is also an isolated singularity of $A_M(V)$.

The Poincaré index for $\mathbf{p}_0$ is η means that if one travels along the boundary of a infinitely small circle around $\mathbf{p}_0$ for one loop, the image of the Gauss map covers the unit circle

$\mathcal{S}$ for η times, counting orientation. From Theorem A.0.3, we know that M induces an automorphism on $\mathcal{S}$. However, a matrix M with a negative determinant will reverse the orientation. Therefore, $\mathbf{p}_0$ has the Poincaré index $sgn(|M|)\eta$.

□

Theorem A.0.4 remains true for higher-dimensional vector fields, although the definition of the Poincaré index and the proof of this theorem require the definition of *Brouwer degree*, which is not covered in this thesis.

Corollary A.0.5. *Matrices in the form* $R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ *maintains the Poincaré indices of the isolated singularities in* V *while* $R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ -\sin(\theta) & -\cos(\theta) \end{pmatrix}$ *negates them.*

Proof. It is straightforward to compute the signs of the determinants of these matrices.

□

It is also interesting to understand the automorphisms induced by these matrices. The first type of matrix induce a translation in $\mathcal{S}$ by θ while the second type of matrices induce a combination of orientation reversal and translation by θ.

REFERENCES

[1] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., and Desbrun, M., "Anisotropic polygonal remeshing," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 485–493, July 2003.

[2] Alliez, P., Meyer, M., and Desbrun, M., "Interactive geometry remeshing," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 347–354, July 2002.

[3] Angenent, S., Haker, S., Tannenbaum, A., and Kikinis, R., "Laplace-beltrami operator and brain surface flattening," *IEEE Transaction on Medical Imaging*, vol. 18, pp. 700–711, 1999.

[4] Asimov, D., "Notes on the topology of vector fields and flows," Tech. Rep. RNR-93-003, NASA Ames Research Center, 1993.

[5] Axen, U. and Edelsbrunner, H., "Auditory morse analysis of triangulated manifolds," *Mathematical Visualization, H.C. Hege and K. Polthier, Eds.*, pp. 223–236, 1998.

[6] Banchoff, T. F., "Critical points and curvature for embedded polyhedral surfaces," *American Mathematical Monthly*, vol. 77, pp. 475–485, 1970.

[7] Benson, D. and Davis, J., "Octree textures," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 785–790, July 2002.

[8] Bremer, P. T., Edelsbrunner, H., Hamann, B., and Pascucci, V., "A multi-resolution data structure for two-dimensional Morse-Smale functions," *Proceeding IEEE Visualization*, pp. 139–146, 2003.

[9] Brostow, G., Essa, I., Steedly, D., and Kwatra, V., "Novel skeletal representation for articulated creatures," *Proceeding of European Conference on Computer Vision*, pp. 66–78, May 2004.

[10] Carr, N. A. and Hart, J. C., "Meshed atlases for real-time procedural solid texturing," *ACM Transactions on Graphics*, vol. 21, no. 2, pp. 106–131, 2002.

[11] Cash, J. R. and Karp, A. H., "A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides," *ACM Transactions on Mathematical Software*, vol. 16, pp. 201–222, 1990.

[12] Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R., "A general method for recovering attribute values on simplified meshes," *Proceeding IEEE Visualization*, pp. 59–66, 1998.

[13] Conley, C., *Isolated Invariant Sets and the Morse Index.* Providence, RI: AMS, 1978. CBMS **38**.

[14] DeBry, D., Gibbs, J., Petty, D. D., and Robins, N., "Painting and rendering textures on unparameterized models," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 763–768, July 2002.

[15] Desbrun, M., Meyer, M., and Alliez, P., "Intrinsic parameterizations of surface meshes," *Proceeding of Eurographics*, pp. 209–218, 2002.

[16] Dey, T. K. and Schipper, H., "A new technique to compute polygonal schema for 2-manifold with application to null-homotopy detection," *Discrete Computational Geometry*, vol. 14, pp. 93–110, 1995.

[17] Dobkin, D. and Kirkpatrick, D., "A linear algorithm for determining the separation of convex polyhedra," *Journal of Algorithms*, vol. 6, pp. 381–392, 1985.

[18] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W., "Multiresolution analysis of arbitrary meshes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1995)*, pp. 173–182, 1995.

[19] Edelsbrunner, H., Harer, J., and Zomorodian, A., "Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds," *In Symp. on Computational Geometry, ACM Press*, pp. 70–79, 2001.

[20] Edelsbrunner, H., Letscher, D., and Zomorodian, A., "Topological persistence and simplification," *Discrete Comput. Geom.*, vol. 28, pp. 511–533, 2002.

[21] Erickson, J. and Har-Peled, S., "Optimally cutting a surface into a disk," *Symposium on Computational Geometry*, pp. 244–253, June 2002.

[22] Floater, M. S., "Parameterization and smooth approximation of surface triangulations," *CAGD*, vol. 14, pp. 231–250, 1997.

[23] Floater, M. S., "Mean value coordinates," *CAGD*, vol. 20, pp. 19–27, 2003.

[24] Floater, M. S., "One-to-one picewise linear mappings over triangulations," *Math. Comp.*, vol. 72, pp. 685–696, 2003.

[25] Floater, M. S. and Hormann, K., "Surface parameterization: a tutorial and survey," *Multiresolution in Geometric Modeling*, 2004. (to appear).

[26] Garland, M., Willmott, A., and Heckbert, P., "Hierarchical face clustering on polygonal surfaces," *Symposium on Interactive 3D Graphics*, pp. 49–58, 2001.

[27] Girshick, A., Interrante, V., Haker, S., and Lemoine, T., "Line direction matters: an argument for the use of principal directions in 3D line drawings," *Proceedings of NPAR*, pp. 43–52, 2000.

[28] Gotsman, C., Gu, X., and Sheffer, A., "Fundamentals of spherical parameterization for 3d meshes," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 358–363, July 2003.

[29] Gottschalk, S., Lin, M. C., and Manocha, D., "OBB-Tree: A hierarchical structure for rapid interference detection," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, pp. 171–180, 1996.

[30] Gu, X., Gortler, S. J., and Hoppe, H., "Geometry images," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 355–361, July 2002.

[31] Guskov, I., Vidimèe, K., Sweldens, W., and Schröder, P., "Normal meshes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pp. 95–102, 2000.

[32] Guskov, I. and Wood, Z., "Topological noise removal," *Graphics Interface*, pp. 19–26, 2001.

[33] Hale, J. and Kocak, H., *Dynamics and Bifurcations*. New York: Springer-Verlag, 1991. Texts in Applied Mathematics **3**.

[34] Hanrahan, P. and Haeberli, P. E., "Direct WYSIWYG painting and texturing on 3d shapes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1990)*, pp. 215–223, 1990.

[35] Hauser, H., Laramee, R. S., and Doleisch, H., "State-of-the-art report 2002 in flow visualization," Tech. Rep. TR-VRVis-2002-003, VRVis Research Center, Vienna, Austria, Jan. 2002.

[36] Hays, J. H. and Essa, I., "Image and video based painterly animation," *NPAR 2004: Third International Symposium on Non-Photorealistic Animation and Rendering*, pp. 113–120, June 2004.

[37] Helman, J. L. and Hesselink, L., "Visualizing vector field topology in fluid flows," *IEEE Computer Graphics and Applications*, vol. 11, pp. 36–46, May 1991.

[38] Hertzmann, A., "Painterly rendering with curved brush strokes of multiple sizes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1998)*, pp. 453–460, 1998.

[39] Hertzmann, A. and Zorin, D., "Illustrating smooth surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pp. 517–526, Aug. 2000.

[40] Hilaga, M., Shinagawa, Y., Kohmura, T., and Kunii, T. L., "Topology matching for fully automatic similarity estimation of 3d shapes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 203–212, 2001.

[41] Hirsch, M. and Smale, S., *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, 1974.

[42] Hoppe, H., "Progressive meshes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, pp. 99–108, 1996.

[43] Hormann, K. and Greiner, G., "MIPS: An efficient global parameterization method," *Curve and Surface Design: Saint-Malo 1999, (edited by Laurent, Sablonnière and Schumaker, Vanderbilt University Press 2000)*, pp. 153–162, 1999.

[44] Kaczynski, T., Mischaikow, K., and Mrozek, M., *Computing Homology*. Springer, 2004. Applied Mathematical Sciences **157**.

[45] Katz, S. and Tal, A., "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 954–961, July 2003.

[46] Khodakovsky, A., Litke, N., and Schröder, P., "Globally smooth parameterizations with low distortion," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 350–357, July 2003.

[47] Kimmel, R. and Sethian, J. A., "Computing geodesic paths on manifolds," *Proceedings of National Academy of Sciences*, vol. 95, pp. 8431–8435, July 1998.

[48] Kindlmann, G. and Weinstein, D., "Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields," *Proceeding IEEE Visualization*, Oct. 1999.

[49] Lazarus, F., Pocchiola, M., Vegter, G., and Veroust, A., "Computing a canonical polygonal schema of an orientable triangulated surface," *17th ACM Symposium on Computational Geometry*, pp. 80–89, June 2001.

[50] Lee, A., Sweldens, W., Schröder, P., Coswar, L., and Dobkin, D., "MAPS: Multiresolution adaptive parameterization of surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1998)*, pp. 95–104, 1998.

[51] Lee, A., Dobkin, D., Sweldens, W., and Schröder, P., "Multiresolution mesh morphing," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1999)*, pp. 343–350, Aug. 1999.

[52] Lee, Y., Kim, H. S., and Lee, S., "Mesh parameterization with a virtual boundary," *Computers & Graphics (Special Issue of the 3rd Israel-Korea Binational Conf. on Geometric Modeling and Computer Graphics)*, vol. 26, no. 5, pp. 677–686, 2002.

[53] Lévy, B., Petitjean, S., Ray, N., and Maillot, J., "Least squares conformal maps for automatic texture atlas generation," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 362–371, July 2002.

[54] Li, X., Woon, T. W., Tan, T. S., and Huang, Z., "Decomposing polygon meshes for interactive applications," *Proceeding of the 2001 symposium on interactive 3D graphics*, pp. 35–42, 2001.

[55] Lindstrom, P. and Turk, G., "Image-driven simplification," *ACM Transactions on Graphics*, vol. 19, no. 3, pp. 204–241, 2000.

[56] Lopes, H., Rossignac, J., Safonova, A., Szymczak, A., and Tavares, G., "Edgebreaker: A simple compression algorithm for surfaces with handles," *Computers and Graphics International Journal*, vol. 27, no. 4, pp. 553–567, 2003.

[57] Maillot, J., Yamia, H., and Verroust, A., "Interactive texture mapping," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1993)*, pp. 27–34, 1993.

[58] Meier, B., "Painterly rendering for animation," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1996)*, pp. 477–484, Aug. 1996.

[59] Milenkovic, V. J., "Rotational polygon containment and minimum enclosure," *Proceedings of the 14th Annual Symposium on Computational Geometry, ACM*, June 1998.

[60] Milnor, J., *Morse Theory*. Princeton, NJ: Annals of Mathematical Studies. Princeton University Press, 1963.

[61] Mischaikow, K., "Topological techniques for efficient rigorous computation in dynamics," *Acta Numerica 2002*, pp. 435–478, 2002. Cambridge University Press.

[62] Mischaikow, K. and Mrozek, M., "Conley index," *Handbook of Dynamic Systems, North-Holland*, vol. 2, pp. 393–460, 2002.

[63] Ni, X., Garland, M., and Hart, J. C., "Fair morse functions for extracting the topological structure of a surface mesh," *ACM Transactions on Graphics (SIGGRAPH 2004)*, vol. 23, Aug. 2004. (to appear).

[64] Okabe, A., Boots, B., and Sugihara, K., *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley & Sons, 1992.

[65] Peachey, D. R., "Solid texturing of complex surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1985)*, pp. 279–286, 1985.

[66] Perlin, K., "An image synthesizer," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1985)*, pp. 287–296, 1985.

[67] Piponi, D. and Borshukov, G., "Seamless texture mapping of subdivision surfaces by model pelting and texture blending," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pp. 471–478, 2000.

[68] Polthier, K. and Schmies, M., "Straightest geodesics on polyhedral surfaces," *Mathematical Visualization, Ed: H.C. Hege, K. Polthier*, pp. 135–150, 1998.

[69] Polthier, K. and Preu, E., "Identifying vector fields singularities using a discrete hodge decomposition," *Mathematical Visualization III, Ed: H.C. Hege, K. Polthier*, pp. 112–134, 2003.

[70] Praun, E., Finkelstein, A., and Hoppe, H., "Lapped textures," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pp. 465–470, Aug. 2000.

[71] Praun, E. and Hoppe, H., "Spherical parametrization and remeshing," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 340–349, July 2003.

[72] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A., "Real-time hatching," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 581–586, Aug. 2001.

[73] Praun, E., Sweldens, W., and Schröder, P., "Consistent mesh parameterizations," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 179–184, July 2001.

[74] Reeb, G., "Sur les points singuliers dune forme de pfaff completement integrable ou dune fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]," *Comptes Randus Acad. Sciences Paris*, vol. 222, pp. 847–849, 1946.

[75] Rockwood, A. and Bunderwala, S., "A toy vector field based on geometric algebra," *Proceeding Application of Geometric Algebra in Computer Science and Engineering, (AGACSE2001)*, pp. 179–185, July 2001.

[76] Rourke, C. and Sanderson, B., *Introduction to Piecewise-Linear Topology.* Springer Verlag, 1972.

[77] Sander, P. V., Gortler, S. J., Snyder, J., and Hoppe, H., "Signal-specialized parameterization," *Proc. 13th Eurographics Workshop on Rendering 2002*, pp. 87–100, 2002.

[78] Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H., "Texture mapping progressive meshes," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 409–416, 2001.

[79] Sander, P. V., Wood, Z. J., Gortler, S. J., Snyder, J., and Hoppe, H., "Multi-chart geometry images," *Proceedings of First Symposium on Geometry Processing 2003*, 2003.

[80] Saroul, L., Gerlach, S., and Hersch, R. D., "Exploring curved anatomic structures with surface sections," *Proceeding IEEE Visualization*, pp. 27–34, 2003.

[81] Scheuermann, G., Krger, H., Menzel, M., and Rockwood, A. P., "Visualizing nonlinear vector field topology," *IEEE Transactions on Visualization and Computer-Grapics*, vol. 4, no. 2, pp. 109–116, 1998.

[82] Sheffer, A. and de Sturler, E., "Parameterization of faceted surfaces for meshing using angle based flattening," *Engineering with Computers*, vol. 17, no. 3, pp. 326–337, 2001.

[83] Sheffer, A. and de Sturler, E., "Smoothing an overlay grid to minimize linear distortion in texture mapping," *ACM Transactions on Graphics*, vol. 21, no. 4, pp. 874–890, 2002.

[84] Sheffer, A. and Hart, J. C., "Seamster: Inconspicuous low-distortion texture seam layout," *Proceeding IEEE Visualization*, pp. 291–298, 2002.

[85] Sorkine, O., Cohen-Or, D., Goldenthal, R., and Lischinski, D., "Bounded-distortion piecewise mesh parameterization," *Proceeding IEEE Visualization*, pp. 355–362, 2002.

[86] Soucy, M., Godin, G., and Rioux, M., "A texture-mapping approach for the compression of colored 3d triangulations," *The Visual Computer*, vol. 12, no. 10, pp. 503–514, 1996.

[87] Stam, J., "Flows on surfaces of arbitrary topology," *ACM Transactions on Graphics (SIGGRAPH 2003)*, July 2003.

[88] Taubin, G., "A signal processing approach to fair surface design," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1995)*, pp. 351–358, 1995.

[89] Theisel, H. and Weinkauf, T., "Vector field metrics based on distance measures of first order critical points," *V. Skala (editor): Journal of WSCG 10*, pp. 121–128, 2002.

[90] Theisel, H., "Designing 2d vector fields of arbitrary topology," *Computer Graphics Forum (Proceedings Eurographics 2002)*, vol. 21, pp. 595–604, July 2002.

[91] Tong, Y., Lombeyda, S., Hirani, A., and Desbrun, M., "Discrete multiscale vector field decomposition," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, pp. 445–452, July 2003.

[92] Tricoche, X., Scheuermann, G., and Hagen, H., "Continous topology simplification of planar vector fields," *Proceeding IEEE Visualization, IEEE Computer Society Press*, pp. 159–166, 2001.

[93] Turk, G., "Texture synthesis on surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 347–354, 2001.

[94] van Wijk, J. J., "Image based flow visualization," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, pp. 745–754, July 2002.

[95] van Wijk, J. J., "Image based flow visualization for curved surfaces," *In: G. Turk, J. van Wijk, R. Moorhead (eds.), Proceedings IEEE Visualization*, pp. 123–130, Oct. 2003.

[96] Vegter, G. and Yap, C. K., "Computational complexity of combinatorial surfaces," *Proc. 6th Annual ACM Symposium on Computational Geometry*, pp. 102–111, 1990.

[97] Wei, L. Y. and Levoy, M., "Texture synthesis over arbitrary manifold surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 355–360, 2001.

[98] Welch, W. and Witkin, A., "Free-form shape design using triangulated surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1994)*, pp. 247–256, 1994.

[99] Westermann, R., Johnson, C., and Ertl, T., "A level-set method for flow visualization," *Proceeding IEEE Visualization*, pp. 147–154, 2000.

[100] Wischgoll, T. and Scheuermann, G., "Detection and visualization of planar closed streamline," *IEEE Transactions on Visualization and Computer Grapics*, vol. 7, no. 2, pp. 165–172, 2001.

[101] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P., "Removing excess topology from isosurfaces," *ACM Transactions on Graphics*, vol. 23, pp. 190–208, Apr. 2004.

[102] Zhang, E. and Turk, G., "Visibility-guided simplification," *Proceeding IEEE Visualization*, pp. 267–274, 2002.